

TUGAS AKHIR - KI141502

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE

DESY NURBAITI RAHMI
NRP 5114 100 030

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Abdul Munif, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

TUGAS AKHIR - KI141502

**IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI
KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE**

DESY NURBAITI RAHMI
NRP 5114 100 030

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Abdul Munif, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESES - KI141502

**IMPLEMENTATION OF ROPE DATA STRUCTURE ON CASE
STUDY IN SPOJ PROBLEM ALPHABETIC ROPE**

DESY NURBAITI RAHMI
NRP 5114 100 030

Supervisor 1
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2
Abdul Munif, S.Kom., M.Sc.

INFORMATICS DEPARTMENT
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

Halaman ini sengaja dikosongkan

**IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI
KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE**

TUGAS AKHIR

**Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada**

**Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember**

Oleh:

**Desy Nurbaiti Rahmi
NRP: 5114 100 030**

Disetujui oleh Dosen Pembimbing Tugas Akhir:

**Rully Soelaiman, S.Kom., M.Kom.
NIP: 197002131994021001**



**Abdul Munif, S.Kom., M.Sc.
NIP: 198608232015041004**

**SURABAYA
JANUARI 2018**

Halaman ini sengaja dikosongkan

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE

Nama : DESY NURBAITI RAHMI
NRP : 5114 100 030
Departemen : Informatika FTIF-ITS
Pembimbing I : Rully Soelaiman, S.Kom., M.Kom.
Pembimbing II : Abdul Munif, S.Kom., M.Sc.

Abstrak

Permasalahan alphabetic rope merupakan sebuah permasalahan yang melibatkan sebuah rentang pencarian. Tipe query secara umum dibagi menjadi dua yaitu, operasi pencarian dan perubahan. Operasi perubahan pada suatu rentang akan menyebabkan perubahan hasil pencarian selanjutnya. Untuk menangani berbagai permasalahan pada operasi alphabetic rope yang harus dilakukan sehingga dibutuhkan struktur data yang mampu mendukung operasi-operasi tersebut dengan efisien.

Pada Tugas Akhir ini akan dirancang penyelesaian permasalahan alphabetic rope antara lain operasi pencarian karakter pada indeks ke-y pada konfigurasi rope saat ini, operasi memotong segmen rope pada indeks ke-x sampai y dan menggabungkan pada bagian depan rope, dan operasi memotong segmen rope pada indeks ke-x sampai y dan menggabungkan pada bagian belakang rope. Struktur data klasik yang biasa digunakan dalam penyelesaian permasalahan ini adalah stuktur data String. Namun penggunaan algoritma String masih kurang efisien dalam hal kecepatan dan kebutuhan memori.

Pada Tugas Akhir ini digunakan struktur data Rope untuk menyelesaikan operasi-operasi tersebut. Hasil uji coba menunjukkan program menghasilkan jalur yang benar dan memiliki pertumbuhan waktu secara logaritmik dengan kompleksitas waktu sebesar $O(\log N)$ per query.

x

Kata Kunci: concatenation, rope, split, string

IMPLEMENTATION OF ROPE DATA STRUCTURE ON CASE STUDY IN SPOJ PROBLEM ALPHABETIC ROPE

Name : DESY NURBAITI RAHMI
NRP : 5114 100 030
Major : Informatics Department Faculty of IT-ITS
Supervisor I : Rully Soelaiman, S.Kom., M.Kom.
Supervisor II : Abdul Munif, S.Kom., M.Sc.

Abstract

Alphabetic rope computation is a problem which involves a specific range of data. Generally it divide by two type of query, range search and update. An update operation would have an impact for the following range search query. To handle various problem in alphabetic rope, an efficient data structure is needed to support the operations.

This undergraduate thesis will be designed problem solving for alphabetic rope query variation such as find the y -th position on current rope, cut the rope segment from x -th to y -th to join at the front of rope and cut the rope segment from x -th to y -th and join at the back of rope. Well known data structures, e.g string are commonly used for solving the this problem. But those algorithm were not efficient enough in the term of running time and memory usage.

In this undergraduate thesis Rope data structure will be used instead for solving those alphabetic rope variations. The program had been tested and proved that it produces correct result and running in $O(\log N)$ per query.

Keywords: concatenation, rope, split, string

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur Penulis panjatkan kepada Allah SWT. atas pimpinan, penyertaan, dan karunia-Nya sehingga Penulis dapat menyelesaikan Tugas Akhir yang berjudul :

IMPLEMENTASI STRUKTUR DATA ROPE PADA STUDI KASUS PERMASALAHAN SPOJ ALPHABETIC ROPE.

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan Penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri Penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama Penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

- Papa, Mama dan keluarga Penulis yang selalu memberikan perhatian, dorongan dan kasih sayang yang menjadi semangat utama bagi diri Penulis sendiri baik selama penulis menempuh masa perkuliahan maupun pengerjaan Tugas Akhir ini.
- Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, pandangan dan bimbingan kepada Penulis baik selama Penulis menempuh masa kuliah maupun selama pengerjaan Tugas Akhir ini.

- Bapak Abdul Munif, S.Kom., M.Sc. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada Penulis.
- Seluruh tenaga pengajar dan karyawan Jurusan Teknik Informatika ITS yang telah memberikan ilmu dan waktunya demi berlangsungnya kegiatan belajar mengajar di Jurusan Teknik Informatika ITS.
- Seluruh teman Penulis di Jurusan Teknik Informatika ITS yang telah memberikan dukungan dan semangat kepada Penulis selama Penulis menyelesaikan Tugas Akhir ini.
- Teman-teman, Kakak-kakak dan Adik-adik *administrator* Laboratorium Arsitektur dan Jaringan Komputer yang selalu menjadi teman untuk berbagi ilmu.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini Penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Januari 2018

Desy Nurbaiti Rahmi

DAFTAR ISI

SAMPUL	i
LEMBAR PENGESAHAN	viii
ABSTRAK	ix
ABSTRACT	xi
KATA PENGANTAR	xiii
DAFTAR ISI	xv
DAFTAR TABEL	xix
DAFTAR GAMBAR	xxi
DAFTAR KODE SUMBER	xxv
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Batasan Masalah	2
1.4 Tujuan	2
1.5 Manfaat	3
1.6 Metodologi	3
1.7 Sistematika Penulisan	4
BAB II DASAR TEORI	5
2.1 Deskripsi Permasalahan	5
2.1.1 Operasi Memotong Segmen <i>Rope</i> dari Indeks ke-X sampai Y dan Menggabungkan pada Bagian Depan <i>Rope</i>	6
2.1.2 Operasi Memotong Segmen <i>Rope</i> dari Indeks ke-X sampai Y dan Menggabungkan pada Bagian Belakang <i>Rope</i>	6
2.1.3 Operasi Mencari Huruf pada Indeks ke-Y dari Konfigurasi <i>Rope</i> Saat Ini	7
2.2 Deskripsi Umum	7

2.2.1	<i>String</i>	8
2.2.2	<i>Struct</i>	8
2.2.3	<i>Binary Search Tree</i>	8
2.2.4	Struktur Data Rope	10
2.2.4.1	<i>Concate</i>	10
2.2.4.2	<i>Split</i>	11
2.2.4.3	Index	12
2.2.5	Perbandingan Rope dengan BST	12
BAB III	DESAIN	15
3.1	Permasalahan <i>Alphabetic Rope</i>	15
3.1.1	Deskripsi Umum Sistem	15
3.1.2	Desain Fungsi Insert	16
3.1.3	Desain Fungsi Build	17
3.1.4	Desain Fungsi Newnode	19
3.1.5	Desain Fungsi Concate	20
3.1.6	Desain Fungsi Split	20
3.1.7	Desain Fungsi Print	21
3.1.8	Desain Fungsi Mutable Begin	23
3.1.9	Desain Fungsi Mutable End	24
BAB IV	IMPLEMENTASI	27
4.1	Lingkungan Implementasi	27
4.2	Rancangan Data	27
4.2.1	Data Masukan	28
4.2.2	Data Keluaran	28
4.3	Implementasi Algoritma	28
4.3.1	<i>Header</i> yang Diperlukan	28
4.3.2	Variabel Global	29
4.3.3	Implementasi Fungsi Main	29
4.3.4	Implementasi Struct Node	30
4.3.5	Implementasi Fungsi Newnode	31
4.3.6	Implementasi Fungsi Build	32
4.3.7	Implementasi Fungsi Getsize	32
4.3.8	Implementasi Fungsi Split	32

4.3.9	Implementasi Fungsi Random	34
4.3.10	Implementasi Fungsi Concate	34
4.3.11	Implementasi Fungsi Insert	36
4.3.12	Implementasi Fungsi Mutable Begin	36
4.3.13	Implementasi Fungsi Mutable End	37
4.3.14	Implementasi Fungsi Print	39
BAB V	UJI COBA DAN EVALUASI	41
5.1	Lingkungan Uji Coba	41
5.2	Uji Coba Kebenaran	41
5.3	Uji Coba Kinerja	48
5.3.1	Operasi 1 Menggabungkan <i>Rope</i> pada Posisi Awal	48
5.3.2	Operasi 2 Menggabungkan <i>Rope</i> pada Posisi Akhir	48
5.3.3	Operasi 3 Mencetak Karakter pada Indeks ke-Y	50
5.4	Analisis Hasil Uji Coba	52
BAB VI	KESIMPULAN	55
6.1	Kesimpulan	55
6.2	Saran	55
DAFTAR PUSTAKA		57
Lampiran A	Hasil Uji Coba Kebenaran pada Situs SPOJ	59
Lampiran B	Hasil Uji Performa Penyelesaian Operasi 1 Menggunakan Struktur Data Rope dan Algorithma String	61
Lampiran C	Hasil Uji Performa Penyelesaian Operasi 2 Menggunakan Struktur Data Rope dan String	63
Lampiran D	Hasil Uji Performa Penyelesaian Operasi 3 Menggunakan Struktur Data Rope dan String	65
BIODATA PENULIS		67

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Tabel 2.1	Tipe Operasi pada Permasalahan <i>Alphabetic Rope</i>	5
Tabel 2.2	Perbandingan Rope dengan String	14
Tabel 3.1	Daftar Variable Untuk Fungsi Main	17
Tabel 3.2	Daftar Variable Untuk Fungsi Insert	18
Tabel 3.3	Daftar Variable Untuk Fungsi Build	19
Tabel 3.4	Daftar Variable Untuk Fungsi Newnode	20
Tabel 3.5	Daftar Variable Untuk Fungsi Concate	21
Tabel 3.6	Daftar Variable Untuk Fungsi Split	22
Tabel 3.7	Daftar Variable Untuk Fungsi Print	23
Tabel 3.8	Daftar Variable Untuk Fungsi Mutable Begin	24
Tabel 3.9	Daftar Variable Untuk Fungsi Mutable End	25
Tabel 5.1	Kecepatan Maksimal, Minimal dan Rata-Rata dari Hasil Uji Coba Sebanyak 15 Kali pada Situs Pengujian SPOJ	47
Tabel B.1	Tabel Hasil Uji Coba pada Operasi 1 dengan Jumlah <i>Query</i> tetap dan Panjang <i>String</i> Bertambah	61
Tabel B.2	Tabel Hasil Uji Coba pada Operasi 1 dengan Panjang <i>String</i> tetap dan Jumlah <i>Query</i> Bertambah	62
Tabel C.1	Tabel Hasil Uji Coba pada Operasi 2 dengan Jumlah <i>Query</i> tetap dan Panjang <i>String</i> Bertambah	63
Tabel C.2	Tabel Hasil Uji Coba pada Operasi 2 dengan Panjang <i>String</i> tetap dan Jumlah <i>Query</i> Bertambah	64

Tabel D.1	Tabel Hasil Uji Coba pada Operasi 3 dengan Jumlah <i>Query</i> tetap dan Panjang <i>String</i> Bertambah	65
Tabel D.2	Tabel Hasil Uji Coba pada Operasi 3 dengan Panjang <i>String</i> tetap dan Jumlah <i>Query</i> Bertambah	66

DAFTAR GAMBAR

Gambar 2.1	Ilustrasi Operasi Memotong Segmen <i>Rope</i> dari Indeks ke-4 sampai 7 dan Menggabungkan pada Bagian Depan <i>Rope</i> .	6
Gambar 2.2	Ilustrasi Operasi Memotong Segmen <i>Rope</i> dari Indeks ke-3 sampai 6 dan Menggabungkan pada Bagian Belakang <i>Rope</i>	7
Gambar 2.3	Ilustrasi Mencari Huruf pada Indeks ke-5 pada Konfigurasi <i>Rope</i> Saat Ini	7
Gambar 2.4	Sebuah <i>Binary Tree</i> dengan Ukuran 9 dan Kedalaman 3, serta 8 Sebagai <i>Root</i>	9
Gambar 2.5	Struktur <i>Rope</i> Berisi <i>String</i> "The_quick_brown_fox"	10
Gambar 2.6	Concate <i>Rope</i> . Setiap <i>Node</i> Berisi Sebuah Karakter dan Berat Masing-Masing <i>Node</i> .	11
Gambar 2.7	Split <i>Rope</i> pada Indeks ke-9	13
Gambar 2.8	Mencari Karakter pada Indeks ke- $i=7$. . .	14
Gambar 3.1	<i>Pseudocode</i> Fungsi Main	16
Gambar 3.2	<i>Pseudocode</i> Fungsi Insert	17
Gambar 3.3	<i>Pseudocode</i> Fungsi Build	18
Gambar 3.4	Build <i>Tree</i> Berdasarkan <i>String</i> "gautambishal". Setiap <i>Node</i> Berisi Sebuah Karakter dan Berat Masing-Masing <i>Node</i>	18
Gambar 3.5	<i>Pseudocode</i> Fungsi Newnode	19
Gambar 3.6	<i>Pseudocode</i> Fungsi Concate	21
Gambar 3.7	<i>Pseudocode</i> Fungsi Split	22
Gambar 3.8	<i>Pseudocode</i> Fungsi Print	23
Gambar 3.9	<i>Pseudocode</i> Fungsi Mutable Begin	24

Gambar 3.10	<i>Pseudocode</i> Fungsi Mutable End	25
Gambar 4.1	Ilustrasi Penyimpanan Hasil Operasi <i>Split Rope</i> pada Struktur Data Pair	33
Gambar 4.2	Ilustrasi Operasi Concate. Setiap <i>Node</i> Berisi Sebuah Karakter dan Nilai Berat Masing-Masing <i>Node</i>	35
Gambar 4.3	Ilustrasi Operasi Mutable Begin. Setiap <i>Node</i> Berisi Sebuah Karakter dan Nilai Prioritas Masing-Masing <i>Node</i>	38
Gambar 4.4	Ilustrasi Operasi Mutable End. Setiap <i>Node</i> Berisi Sebuah Karakter dan Nilai Prioritas Masing-Masing <i>Node</i>	40
Gambar 5.1	Contoh kasus uji permasalahan Alphabetic Rope	42
Gambar 5.2	Pembentukan <i>Root Rope</i>	42
Gambar 5.3	Pembentukan <i>Rope</i> pada Tingkat ke-2	43
Gambar 5.4	Pembentukan <i>Rope</i> pada Tingkat ke-3	43
Gambar 5.5	Struktur <i>Rope</i> yang Terbentuk	44
Gambar 5.6	Konfigurasi <i>Rope</i> Setelah Operasi 2 0 5	45
Gambar 5.7	Hasil Luaran Program pada Contoh Kasus Uji Alphabetic Rope	46
Gambar 5.8	Hasil Uji Coba pada Situs Penilaian SPOJ	46
Gambar 5.9	Hasil Uji Coba pada Situs Penilaian SPOJ	47
Gambar 5.10	Hasil Uji Coba pada Operasi 1 dengan Jumlah <i>Query</i> Tetap dan Panjang <i>String</i> Bertambah	49
Gambar 5.11	Hasil Uji Coba pada Operasi 1 dengan Jumlah <i>String</i> Tetap dan Jumlah <i>Query</i> Bertambah	49
Gambar 5.12	Hasil Uji Coba pada Operasi 2 dengan Jumlah <i>Query</i> Tetap dan Jumlah <i>String</i> Bertambah	50

Gambar 5.13 Hasil Uji Coba pada Operasi 2 dengan Panjang <i>String</i> Tetap dan Jumlah <i>Query</i> Bertambah	51
Gambar 5.14 Hasil Uji Coba pada Operasi 3 dengan Jumlah <i>Query</i> Tetap dan Panjang <i>String</i> Bertambah	51
Gambar 5.15 Hasil Uji Coba pada Operasi 3 dengan Jumlah <i>String</i> Tetap dan Jumlah <i>Query</i> Bertambah	52
Gambar A.1 Hasil Pengujian Sebanyak 15 Kali pada Situs Penilaian Daring SPOJ	59

Halaman ini sengaja dikosongkan

DAFTAR KODE SUMBER

Kode Sumber 2.1	Struktur Struct Node	8
Kode Sumber 4.1	<i>Header</i> yang diperlukan	29
Kode Sumber 4.2	Variabel Global	29
Kode Sumber 4.3	Fungsi Main	30
Kode Sumber 4.4	Fungsi Struct Node	30
Kode Sumber 4.5	Fungsi Newnode	31
Kode Sumber 4.6	Fungsi Build	32
Kode Sumber 4.7	Fungsi Getsize	32
Kode Sumber 4.8	Fungsi Split(1)	33
Kode Sumber 4.9	Fungsi Split(2)	34
Kode Sumber 4.10	Fungsi Random	34
Kode Sumber 4.11	Fungsi Concate(1)	35
Kode Sumber 4.12	Fungsi Concate(2)	36
Kode Sumber 4.13	Fungsi Insert	36
Kode Sumber 4.14	Fungsi Mutable Begin	37
Kode Sumber 4.15	Fungsi Mutable End	39
Kode Sumber 4.16	Fungsi Print	39

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Di era modern ini, sudah banyak algoritma dan struktur data yang ditemukan. Masing-masing algoritma seringkali membutuhkan struktur data yang tepat untuk membantu menyelesaikan permasalahan tersebut. Salah satunya pada permasalahan *string*.

Topik Tugas Akhir ini mengacu pada permasalahan *string* pada permasalahan klasik *Alphabetic Rope*[1]. Pada permasalahan ini terdapat sekumpulan data berupa *string* yang diberikan di awal. Dari sekumpulan *string* ini akan dilakukan sejumlah pertanyaan dan perubahan atau *update* pada data awal yang diberikan. Struktur data yang diduga dapat membantu penyelesaian permasalahan ini adalah struktur data Rope.

Hasil Tugas Akhir ini diharapkan dapat memberi gambaran mengenai performa dari struktur data Rope dalam penyelesaian permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana menganalisis serta menentukan algoritma dan struktur data yang tepat untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dengan optimal?
2. Bagaimana hasil perbandingan performa dari struktur data *Rope* yang dibangun dalam menjawab permasalahan klasik *Alphabetic Rope* yang diberikan?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas jumlah maksimum data awal yang diberikan adalah 100.000 karakter.
3. Batas maksimum jumlah *query* dan/atau *update* adalah 100.000 perintah.
4. Batas waktu eksekusi program adalah 1 detik.
5. Dataset yang digunakan adalah dataset pada permasalahan klasik *Alphabetic Rope*.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Melakukan analisis dan mendesain algoritma dan struktur data untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dengan pembuktian (*proofing*) yang jelas.
2. Membandingkan performa dari algoritma dan struktur data yang dapat digunakan untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dengan analisis yang jelas.

1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui penggunaan struktur data yang tepat untuk menyelesaikan permasalahan klasik *Alphabetic Rope* secara optimal dan efisien.

1.6 Metodologi

Metodologi yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir
Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir yang berisi gagasan untuk menyelesaikan permasalahan *rope* pada studi kasus permasalahan klasik *Alphabetic Rope*.
2. Studi literatur
Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai pengetahuan atau metode yang dapat digunakan dalam penyelesaian masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma yang digunakan untuk penyelesaian permasalahan ini, materi-materi tersebut didapatkan dari buku, jurnal, maupun internet.
3. Desain
Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan permasalahan klasik *Alphabetic Rope*.
4. Implementasi perangkat lunak
Pada tahap ini dilakukan implementasi atau realiasi dari rancangan desain algoritma yang telah dibangun pada tahap desain ke dalam bentuk program.
5. Uji coba dan evaluasi

Pada tahap ini dilakukan uji coba kebenaran implementasi. Pengujian kebenaran dilakukan pada sistem penilaian daring SPOJ sesuai dengan masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai.

6. Penyusunan buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir ini:

1. BAB I: PENDAHULUAN

Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan Tugas Akhir.

2. BAB II: DASAR TEORI

Bab ini berisi dasar teori mengenai permasalahan dan algoritma penyelesaian yang digunakan dalam Tugas Akhir

3. BAB III: DESAIN

Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.

4. BAB IV: IMPLEMENTASI

Bab ini berisi implementasi berdasarkan desain algoritma yang telah dilakukan pada tahap desain.

5. BAB V: UJI COBA DAN EVALUASI

Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.

6. BAB VI: PENUTUP

Bab ini berisi kesimpulan dan saran yang didapat dari hasil uji coba yang telah dilakukan.

BAB II

DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan Tugas Akhir ini.

2.1 Deskripsi Permasalahan

Terdapat sebuah *alphabetic rope*[1] yang terdiri dari karakter-karakter yang terlihat seperti serangkaian *string* dengan karakter yang terdiri dari huruf kecil. Diberikan beberapa operasi perubahan dan atau pertanyaan. Tabel 2.1 adalah macam-macam tipe operasi pada permasalahan klasik *Alphabetic Rope*.

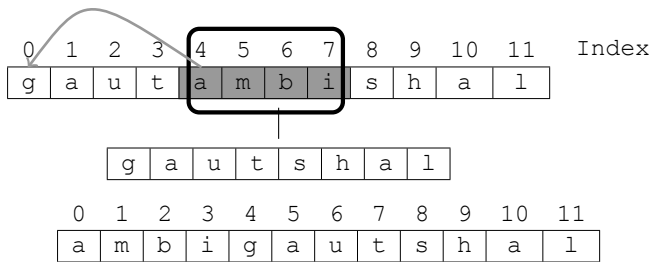
Tabel 2.1 Tipe Operasi pada Permasalahan *Alphabetic Rope*

1 X Y	Memotong segmen <i>rope</i> dari indeks ke-X sampai Y dan menggabungkan pada bagian depan <i>rope</i>
2 X Y	Memotong segmen <i>rope</i> dari indeks ke-X sampai Y dan menggabungkan pada bagian belakang <i>rope</i>
3 Y	Mencetak pada baris baru huruf pada indeks ke-Y dari konfigurasi <i>rope</i> saat ini

Mula-mula terdapat sejumlah N data berupa *string* sebagai data awal. Kemudian terdapat serangkaian Q buah operasi yang akan dilakukan pada data awal yang telah diberikan. Sebuah operasi dapat berupa sebuah tindakan yang mengubah kondisi data. Variasi dari operasi-operasi yang menjadi permasalahan pada Tugas Akhir ini akan dijelaskan pada subbab-subbab di bawah.

2.1.1 Operasi Memotong Segmen *Rope* dari Indeks ke-X sampai Y dan Menggabungkan pada Bagian Depan *Rope*

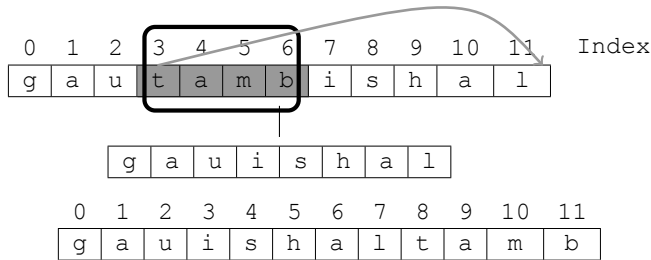
Terdapat 3 parameter pada operasi ini yaitu *type*, *x*, dan *y* yang berupa posisi dari indeks dari *string* yang akan dipotong dan digabungkan pada posisi awal *rope*. Misalkan terdapat *string* "gautambishal", dilakukan operasi 1 pada indeks ke-4 sampai 7. Maka *string* pada indeks tersebut akan dipotong dan digabungkan pada posisi awal dari *rope*. Ilustrasi permasalahan ini dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ilustrasi Operasi Memotong Segmen *Rope* dari Indeks ke-4 sampai 7 dan Menggabungkan pada Bagian Depan *Rope*

2.1.2 Operasi Memotong Segmen *Rope* dari Indeks ke-X sampai Y dan Menggabungkan pada Bagian Belakang *Rope*

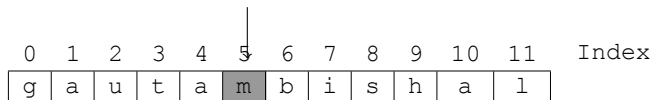
Sama dengan operasi pada subbab 2.1.1, operasi ini memiliki 3 parameter yaitu *type*, *x* dan *y*. Namun potongan *string* akan digabungkan pada posisi akhir *rope*. Misal terdapat *string* "gautambishal", dan dilakukan operasi 2 pada indeks ke-3 sampai 6. Maka *string* pada indeks tersebut akan digabungkan pada bagian akhir dari *rope*. Ilustrasi permasalahan ini dapat dilihat pada Gambar 2.2.



Gambar 2.2 Ilustrasi Operasi Memotong Segmen *Rope* dari Indeks ke-3 sampai 6 dan Menggabungkan pada Bagian Belakang *Rope*

2.1.3 Operasi Mencari Huruf pada Indeks ke-Y dari Konfigurasi *Rope* Saat Ini

Terdapat dua buah parameter pada operasi ini yaitu *type* dan *y*. Nilai *y* merupakan posisi indeks huruf yang akan dicetak. Misalkan terdapat sebuah *rope* yang berisi string "gautambishal", dilakukan operasi 3 pada indeks ke-5. Maka jawaban dari pertanyaan ini adalah *m*. Ilustrasi operasi ini dapat dilihat pada Gambar 2.3.



Gambar 2.3 Ilustrasi Mencari Huruf pada Indeks ke-5 pada Konfigurasi *Rope* Saat Ini

2.2 Deskripsi Umum

Pada subbab ini akan dijelaskan mengenai deskripsi-deskripsi umum yang terdapat pada Tugas Akhir ini.

2.2.1 *String*

Pada dunia ilmu komputer, *string* didefinisikan sebagai sebuah rangkaian karakter, baik sebagai konstanta literal atau semacam sebuah variabel. Sebagai variabel, *string* dapat dimungkinkan unsur-unsurnya dapat dimutasi dan panjangnya menjadi berubah. *String* pada umumnya dipahami sebagai sebuah struktur data dan diimplementasikan menggunakan struktur data Array[2].

2.2.2 *Struct*

Struct merupakan suatu struktur data yang menggabungkan sekumpulan data yang memiliki tipe data yang berbeda. Dalam penerapannya, *struct* digunakan untuk membuat *node* yang berfungsi menyimpan informasi-informasi berupa karakter, berat *node*, *pointer node* dan konstruktor. Kode Sumber 2.1 adalah struktur *Struct* yang digunakan dalam implementasi *rope*.

```

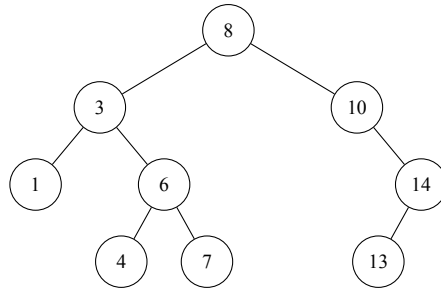
1  struct Node {
2      Node *left, *right;
3      int size;
4      char value;
5      Node(char v);
6      Node* update();
7  };

```

Kode Sumber 2.1 Struktur Struct Node

2.2.3 *Binary Search Tree*

Sebuah konsep penyimpanan data, dimana data disimpan dalam bentuk *tree* yang setiap *node* dapat memiliki anak maksimal 2 *node*. BST(*Binary Search Tree*) merupakan susunan *tree* yang sudah terurut. Pencarian dan *insert* data pada BST dapat dilakukan dengan mudah dan memiliki rata-rata kompleksitas sebesar $O(\log n)$, dan pada kondisi terburuknya dimana BST tidak berimbang dan



Gambar 2.4 Sebuah *Binary Tree* dengan Ukuran 9 dan Kedalaman 3, serta 8 Sebagai *Root*

berbentuk seperti *linked list* memerlukan waktu sebesar $O(n)[3]$. Ilustrasi *binary tree* dapat dilihat pada Gambar 2.4.

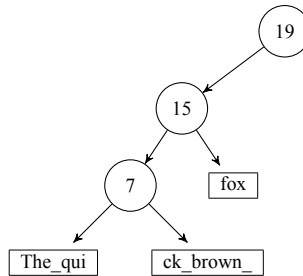
Theorema 2.1

Jika R adalah sebuah *tree* dengan berukuran N maka R adalah sebuah *complete binary tree* dengan tinggi $H(R) = O(\log N)[3]$.

Proof Misalkan $H(N)$ menyatakan ketinggian *tree* yang berukuran N , akan terdapat N buah *node* pada *tree*. Misalkan *node* 8 terpilih menjadi *root*, maka semua *node* dengan nilai kurang dari *root* akan ditempatkan sebagai *subtree* anak kiri, dan yang lebih besar akan di tempatkan sebagai *subtree* anak kanan. Kemudian masing-masing *subtree* kiri dan kanan dari *node* 8 akan mengikuti pola yang sama secara rekursif. Jadi, ketinggian dari *tree* adalah 1 ditambah ketinggian maksimal dari *subtree* kiri dan *subtree* kanan. Maka,

$$H(N) = \begin{cases} |N|, & |N| \leq 1. \\ \frac{1}{|N|} \sum_{a \in N} (1 + \max(H[N_{<a}], H[N_{>a}])), & |N| > 1. \end{cases}$$

Dengan *master theorm* maka rekurens ini terselesaikan menjadi $H(\log N)$.



Gambar 2.5 Struktur *Rope* Berisi *String* "The_quick_brown_fox"

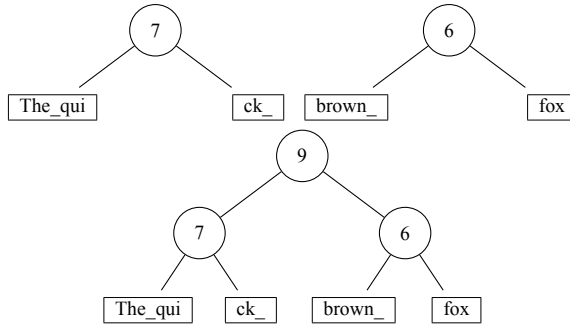
2.2.4 Struktur Data *Rope*

Pada permasalahan klasik *Alphabetic Rope*, penyelesaian menggunakan struktur data *String* dirasa tidak cukup menyelesaikan permasalahan dimana rata-rata kompleksitas yang dimiliki sebesar $O(n)$ [3]. Solusi ini masih kurang efisien untuk mengatasi batasan masukan yang cukup besar pada setiap dataset.

Penyelesaian permasalahan yang ditawarkan adalah dengan menggunakan struktur *Rope*. Struktur data *Rope* adalah sebuah struktur *binary tree* yang setiap *leaf node*nya berisi *string* yang pendek. Setiap *node* berisi nilai bobot yang sama dengan jumlah panjang *string* pada anak kirinya. Dengan demikian, *string* yang diwakilkan pada sebuah *tree* adalah gabungan dari *leaf node*nya dari kiri ke kanan, seperti pada Gambar 2.5. Struktur data *Rope* memiliki beberapa operasi yang dapat dilakukan antara lain *concat*, *insert*, *split*, *delete* dan *report*[5].

2.2.4.1 *Concat*

Operasi $\text{Concat}(R_1, R_2)$ menggabungkan dua buah *rope*, R_1 dan R_2 menjadi sebuah *rope* utuh. Penggabungan kedua *rope* R_1 dan R_2 dilakukan dengan membuat sebuah *node* baru dengan anak kiri



Gambar 2.6 Concat *Rope*. Setiap *Node* Berisi Sebuah Karakter dan Berat Masing-Masing *Node*

R_1 dan anak kanan R_2 . Berat dari *parent node* merupakan panjang dari anak kiri R_1 , dan memerlukan kompleksitas waktu sebesar $O(\log N)$ jika *tree* seimbang[5]. Ilustrasinya dapat dilihat pada Gambar 2.6.

2.2.4.2 Split

Operasi $\text{split}(R, i)$ membagi *rope* R menjadi dua buah *rope* baru R_1 dan R_2 , dimana $R_1 = C_1, \dots, C_i$ dan $R_2 = C_{i+1}, \dots, C_n$. Terdapat dua permasalahan yang harus ditangani.

1. Memotong pada bagian akhir *string* (karakter terakhir pada *leaf node*)
2. Memotong pada bagian tengah sebuah *string*

Permasalahan kedua dengan membuat sebuah *node* baru membagi *string* pada titik *split*, lalu membuat *node* baru yang merupakan induk dari kedua komponen.

Sebagai contoh, *rope* yang berisi 19 karakter yang digambarkan pada Gambar 2.7 akan dibagi pada indeks ke- $i = 9$. Langkah pertama mencari *node* pada indeks ke- $i = 9$. Dikarenakan posisi i

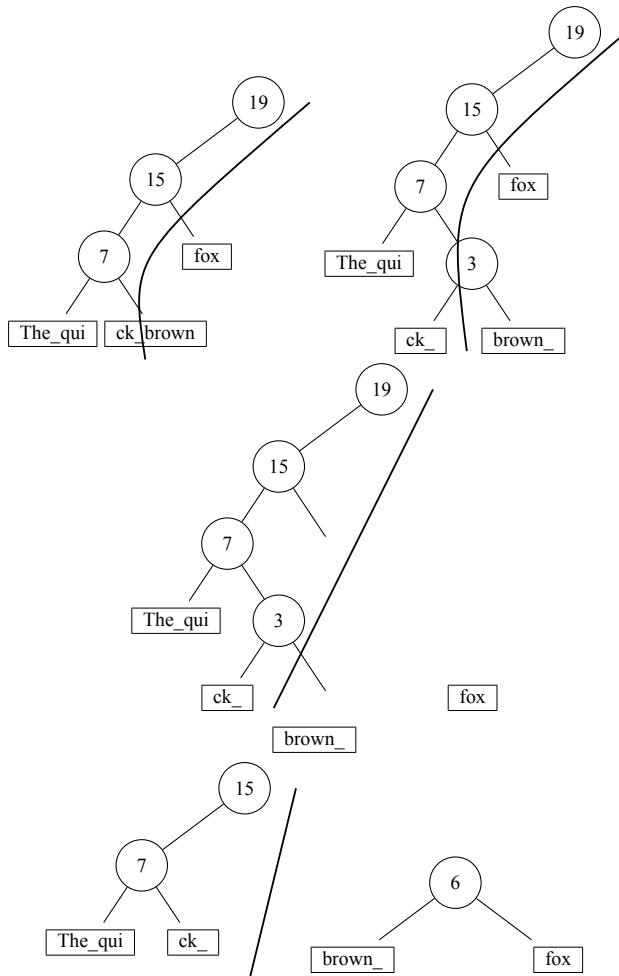
merupakan bagian tengah dari *string* maka dibuatkan sebuah *node* baru menyimpan masing-masing potongan *string*. Kemudian hapus sambungan *node* "*brown_*" dengan *parentnya*. Selanjutnya hapus sambungan *node* "*fox*" dengan *parentnya*. Dan dilakukan *update* berat masing-masing *node*. Untuk setiap *node* yang terpisah dari *rope* utama akan dibangun kembali dengan membuat sebuah *node* baru sebagai penghubung.

2.2.4.3 Index

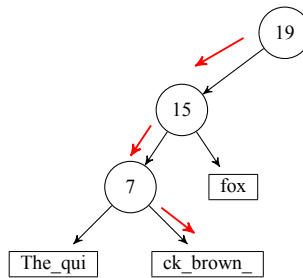
Operasi $\text{index}(i)$ menghasilkan sebuah karakter dari *rope* pada posisi ke- i . Sebagai contoh, untuk mencari karakter pada indeks ke- $i=8$ dari *rope* pada Gambar 2.8, dimulai dari *root*, didapatkan 19 lebih besar dari 8 dan terdapat anak kiri, maka pencarian dilanjutkan ke anak kiri. 15 lebih besar dari 8, pencarian dilanjutkan ke anak kiri. Selanjutnya 7 lebih kecil dari 8 pencarian dilanjutkan ke anak kanan dan menyisakan nilai $i = 1$. Karena sudah berada di *leaf node*, maka karakter pada indeks ke-1 dari *string* "*ck_brown_*" jawabannya.

2.2.5 Perbandingan Rope dengan BST

Secara struktur, *rope* merupakan sebuah *binary tree*. Namun memiliki beberapa perbedaan yang dapat dilihat pada Tabel 2.2.



Gambar 2.7 Split *Rope* pada Indeks ke-9

Gambar 2.8 Mencari Karakter pada Indeks ke- $i=7$

Tabel 2.2 Perbandingan Rope dengan String

Rope	BST
Struktur: sebuah <i>tree</i>	Struktur: sebuah <i>tree</i>
Operasi: Insert, Concat, Split, Delete, Report	Operasi: Searching, Insert, Delete, Traversal
Insert: dilakukan sesuai dengan urutan <i>string</i>	Insert: dilakukan dengan meletakkan setiap <i>node</i> yang bernilai lebih kecil dari nilai <i>node</i> saat ini pada posisi kiri dan yang lebih besar pada posisi kanan
Delete: dilakukan dengan dua buah operasi <i>split</i> dan sebuah <i>concat</i>	Delete: dilakukan dengan menghapus <i>node</i> yang diinginkan dan mengganti dengan <i>node</i> anaknya

BAB III

DESAIN

Pada bab ini akan dibahas tentang desain dan algoritma untuk menyelesaikan permasalahan-permasalahan pada Tugas Akhir ini.

3.1 Permasalahan *Alphabetic Rope*

Pada permasalahan ini sistem akan dibangun untuk menyelesaikan tiga jenis operasi yaitu, memotong segmen *rope* dari indeks ke- x sampai y dan menggabungkan pada bagian depan *rope*, memotong segmen *rope* dari indeks ke- x sampai y dan menggabungkan pada bagian belakang *rope* dan mencetak huruf pada indeks ke- y dari konfigurasi *rope* saat ini.

3.1.1 Deskripsi Umum Sistem

Program akan diawali dengan menerima masukan berupa *string* sebagai data awal dan banyak data uji. Kemudian akan dilakukan proses Insert dan membuah sebuah *rope* yang berisi *string* masukan. Untuk setiap data uji berupa sebuah baris yang diawali dengan data masukan berupa tipe operasi yang akan dilakukan. Setiap data uji, selain tipe operasi 3 diikuti dengan dua data masukan lain yaitu x dan y yang dipisahkan oleh sebuah spasi. Pada data masukan dengan tipe operasi 1 dan 2, nilai x dan y merupakan indeks posisi awal dan indeks posisi akhir dari *string* pada *rope* yang akan diolah. Proses ini memanfaatkan dua buah operasi Split dan dua buah operasi Concat yang dilakukan dengan memanggil fungsi `Mutable_end` atau `Mutable_begin`. Sedangkan pada tipe operasi 3, data masukan lain hanya berupa y yang merupakan indeks dari huruf

Masukan	<i>String</i> , Banyak data uji Q , Operasi ($type, x, y$)
Keluaran	Karakter pada indeks ke- y pada tipe operasi 3
<pre> Main() 1 $r = \text{insert}(r, st)$ 2 while $Q \neq \emptyset$ 3 if $type = 3$ 4 $\text{print}(r, y)$ 5 else 6 if $type = 1$ 7 $r = \text{mutable_begin}(r, x, y - x + 1)$ 8 else 9 $r = \text{mutable_end}(r, x, y - x + 1)$ </pre>	

Gambar 3.1 *Pseudocode* Fungsi Main

yang dicari pada konfigurasi *rope* saat ini. Hasil dari pencarian data pada *rope* akan ditampilkan di layar, yang dilakukan dengan memanggil fungsi Print. Secara garis besar, Fungsi Main dapat dilihat pada Gambar 3.1.

Daftar variabel yang digunakan pada program utama dapat dilihat pada Tabel 3.1.

3.1.2 Desain Fungsi Insert

Fungsi Insert digunakan untuk memasukkan *string* ke dalam *rope*. *Pseudocode* Fungsi Insert ditunjukkan pada Gambar 3.2.

Daftar variabel yang digunakan pada fungsi Insert dapat dilihat pada Tabel 3.2.

Tabel 3.1 Daftar Variable Untuk Fungsi Main

No	Nama Variable	Tipe Data	Keterangan
1	st	Array of Char	String masukan awal
2	r	Node*	Inisialisasi awal <i>root</i>
3	Q	Integer	Jumlah data uji
4	type	Integer	Tipe operasi
5	x	Integer	Posisi indeks
6	y	Integer	Posisi indeks

Masukan	<i>root rope, string</i>
Keluaran	Fungsi <code>concat(<i>r, x</i>)</code>
<code>Insert(<i>r, str</i>)</code> <code>1 <i>x</i> = build(<i>str, str</i> + <code>strlen(<i>str</i>)</code>)</code> <code>2 return concat(<i>r, x</i>)</code>	

Gambar 3.2 Pseudocode Fungsi Insert

3.1.3 Desain Fungsi Build

Fungsi Build digunakan ketika melakukan Insert ke dalam *rope*. Hal ini bertujuan untuk membuat struktur *tree* dari *rope*. Pembuatan *tree* pada *rope* ini akan dilakukan dari indeks *string* yang berada di posisi tengah, sampai pada ujung-ujungnya. Setiap *node* akan memiliki anak kiri dan anak kanan jika dan hanya jika masih terdapat *string* yang tersisa. Ilustrasi pembuatan *rope* dapat dilihat pada Gambar 3.4. Pseudocode Fungsi Build ditunjukkan pada Gambar 3.3.

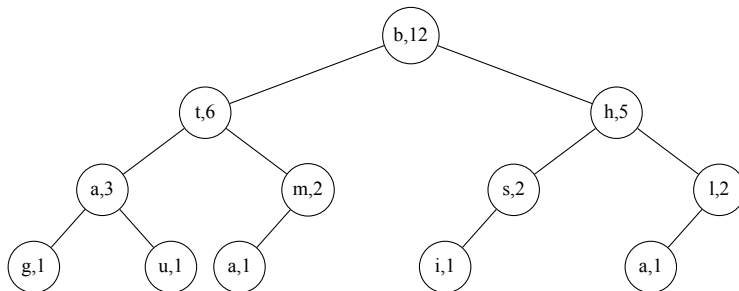
Daftar variabel yang digunakan pada fungsi Build dapat dilihat pada Tabel 3.3.

Tabel 3.2 Daftar Variable Untuk Fungsi Insert

No	Nama Variable	Tipe Data	Keterangan
1	str	Array of Char	<i>String</i> masukan awal
2	r	Node*	Inisialisasi awal <i>root</i>
3	x	Node*	Inisialisasi node penyimpanan hasil <i>build</i>

Masukan	<i>string</i> awal, <i>string</i> awal
Keluaran	Fungsi newnode(* <i>mid</i> , build(<i>st</i> , <i>mid</i>), build(<i>mid</i> + 1, <i>en</i>))
Build(* <i>st</i> , * <i>en</i>) 1 if <i>st</i> = <i>en</i> return NULL 2 * <i>mid</i> = (<i>st</i> + <i>en</i> - <i>st</i>)/2 3 return newnode(* <i>mid</i> , build(<i>st</i> , <i>mid</i>), build(<i>mid</i> +1, <i>en</i>))	

Gambar 3.3 Pseudocode Fungsi Build



Gambar 3.4 Build Tree Berdasarkan String "gautambishal". Setiap Node Berisi Sebuah Karakter dan Berat Masing-Masing Node

Tabel 3.3 Daftar Variable Untuk Fungsi Build

No	Nama Variable	Tipe Data	Keterangan
1	st	Char*	Karakter pada indeks awal
2	en	Char*	Karakter pada indeks akhir
3	mid	Char*	Karakter pada indeks tengah

Masukan	<i>string, pointer kiri, pointer kanan</i>
Keluaran	Sebuah <i>rope</i>
<pre> Newnode(<i>ch, left, right</i>) 1 <i>r</i> = new Node(<i>ch</i>) 2 <i>r.left</i> = <i>left</i> 3 <i>r.right</i> = <i>right</i> 4 <i>r.update</i>() 5 return <i>r</i> </pre>	

Gambar 3.5 *Pseudocode* Fungsi Newnode

3.1.4 Desain Fungsi Newnode

Fungsi Newnode digunakan untuk membentuk sebuah *node* yang berisikan karakter yang diberikan dan relasi terhadap karakter pada posisi yang bersebelahan. Fungsi ini dijalankan setelah fungsi Build dipanggil. *Pseudocode* fungsi Newnode ditunjukkan pada Gambar 3.5.

Daftar variabel yang digunakan pada fungsi Newnode dapat dilihat pada Tabel 3.4.

Tabel 3.4 Daftar Variable Untuk Fungsi Newnode

No	Nama Variable	Tipe Data	Keterangan
1	ch	Char*	Karakter masukan
2	left	Node*	Pointer menuju anak kiri
3	right	Node*	Pointer menuju anak kanan
4	r	Node*	Node penyimpan data masukan

3.1.5 Desain Fungsi Concat

Fungsi Concat bertujuan untuk menggabungkan dua buah *rope* menjadi sebuah *rope* utuh. Penggabungan dua buah *rope* R_1 dan R_2 dilakukan dengan menghubungkan *root* R_1 dengan R_2 tergantung pada penggunaan nilai acak yang didapatkan berdasarkan hasil modular nilai acak dari kedua *root*. Apabila hasil modular nilai acak lebih kecil dari berat *root* R_1 maka *rope* R_2 akan menjadi anak kanan dari *rope* yang baru terbentuk. Sebaliknya jika mendapatkan hasil modular nilai acak lebih besar atau sama dengan berat *root* R_1 maka R_2 akan menjadi *root* dari *rope* yang baru terbentuk dan *rope* R_1 menjadi anak kirinya. *Pseudocode* Fungsi Concat ditunjukkan pada Gambar 3.6.

Daftar variabel yang digunakan pada fungsi Concat dapat dilihat pada Tabel 3.5.

3.1.6 Desain Fungsi Split

Fungsi Split bertujuan untuk membagi *rope* menjadi dua buah *rope* baru R_1 dan R_2 . Terdapat sebuah parameter i yang merupakan posisi indeks *rope* yang akan dibagi. R_1 berisi semua *node* dengan indeks lebih kecil dari parameter. Sedangkan R_2 berisi semua *node*

Masukan	<i>rope</i> yang akan digabungkan
Keluaran	Sebuah <i>rope</i> utuh
<pre> Concat(R_1, R_2) 1 if ($!R_1 !R_2$) return $R_1 ? R_1 : R_2$ 2 if ($\text{random}(R_1.\text{size}, R_2.\text{size})$) 3 $R_1.\text{right} = \text{concat}(R_1.\text{right}, R_2)$ 4 return $R_1.\text{update}()$ 5 else 6 $R_2.\text{left} = \text{concat}(R_1, R_2.\text{left})$ 7 return $R_2.\text{update}()$ </pre>	

Gambar 3.6 *Pseudocode* Fungsi Concat

Tabel 3.5 Daftar Variable Untuk Fungsi Concat

No	Nama Variable	Tipe Data	Keterangan
1	R_1	Node*	Pointer menuju <i>rope</i> pertama
2	R_2	Node*	Pointer menuju <i>rope</i> kedua

dengan indeks lebih besar sama dengan parameter. Operasi ini memanfaatkan struktur data Pair yang berupa pasangan dari dua *pointer* menuju *node*. *Pseudocode* Fungsi Split ditunjukkan pada Gambar 3.7.

Daftar variabel yang digunakan pada fungsi Split dapat dilihat pada Tabel 3.6.

3.1.7 Desain Fungsi Print

Fungsi Print bertujuan untuk mendapatkan karakter pada suatu indeks di dalam *rope*. Fungsi ini digunakan untuk menjawab pertanyaan pada operasi dengan *type* = 3. *Pseudocode* Fungsi

Masukan	<i>rope</i> , posisi <i>split</i>
Keluaran	Dua buah <i>rope</i> baru
<pre> Split(<i>r</i>, <i>pos</i>) 1 if !<i>r</i> return make_pair(<i>R</i>₁, <i>R</i>₂) 2 <i>idx</i> = getSize(<i>r.left</i>) 3 if <i>idx</i> < <i>pos</i> 4 <i>temp</i> = split(<i>r.right</i>, <i>pos</i> - <i>idx</i> - 1) 5 <i>r.right</i> = <i>temp.first</i> 6 <i>R</i>₂ = <i>temp.second</i> 7 <i>R</i>₁ = <i>r</i> 8 else 9 <i>temp</i> = split(<i>r.left</i>, <i>pos</i>) 10 <i>R</i>₁ = <i>temp.first</i> 11 <i>r.left</i> = <i>temp.second</i> 12 <i>R</i>₂ = <i>r</i> 13 <i>r</i>.update() 14 return make_pair(<i>R</i>₁, <i>R</i>₂) </pre>	

Gambar 3.7 Pseudocode Fungsi Split

Tabel 3.6 Daftar Variable Untuk Fungsi Split

No	Nama Variable	Tipe Data	Keterangan
1	<i>r</i>	Node*	<i>Rope</i> yang akan displit
2	<i>pos</i>	Integer	Posisi indeks <i>split</i>
3	<i>R</i> ₁	Node*	Pointer menuju <i>rope</i> pertama
4	<i>R</i> ₂	Node*	Pointer menuju <i>rope</i> kedua
5	<i>idx</i>	Integer	Penyimpan nilai berat anak kiri
6	<i>temp</i>	Pair Node*	Penyimpan hasil <i>split</i>

Masukan	<i>rope</i> , posisi indeks
Keluaran	Sebuah karakter pada indeks ke- <i>y</i> dari <i>rope</i>
Print(<i>r</i> , <i>pos</i>) 1 <i>idx</i> = getSize(<i>r.left</i>) 2 if (<i>idx</i> = <i>pos</i>) return <i>r.value</i> 3 if (<i>pos</i> ≤ <i>idx</i>) return print(<i>r.left</i> , <i>pos</i>) 4 else return print(<i>r.right</i> , <i>pos</i> − <i>idx</i> − 1)	

Gambar 3.8 *Pseudocode* Fungsi Print

Print ditunjukkan pada Gambar 3.8.

Daftar variabel yang digunakan pada fungsi Print dapat dilihat pada Tabel 3.7.

Tabel 3.7 Daftar Variable Untuk Fungsi Print

No	Nama Variable	Tipe Data	Keterangan
1	r	Node*	<i>Rope</i> yang akan digunakan
2	pos	Integer	Posisi indeks dari karakter yang dicari
3	idx	Integer	Penyimpan nilai berat anak kiri

3.1.8 Desain Fungsi Mutable Begin

Fungsi Mutable Begin digunakan untuk menjawab pertanyaan dengan *type* = 1. Hal ini bertujuan untuk menggabungkan hasil potongan *rope* pada bagian depan *rope*. Fungsi ini memanfaatkan dua buah operasi *split* dan dua buah operasi *concat*. *Pseudocode* Fungsi Mutable Begin ditunjukkan pada Gambar 3.9.

Daftar variabel yang digunakan pada fungsi Mutable Begin dapat

Masukan	<i>rope</i> , posisi indeks, panjang <i>string</i>
Keluaran	Fungsi <code>concat(<i>sec.first</i>, concat(<i>fir.first</i>, <i>sec.second</i>))</code>
Print(<i>r, pos</i>) 1 <i>fir</i> = split(<i>x, pos</i>) 2 <i>sec</i> = split(<i>fir.second, len</i>) 3 return concat(<i>sec.first</i> , concat(<i>fir.first, sec.second</i>))	

Gambar 3.9 *Pseudocode* Fungsi Mutable Begin

dilihat pada Tabel 3.8.

Tabel 3.8 Daftar Variable Untuk Fungsi Mutable Begin

No	Nama Variable	Tipe Data	Keterangan
1	r	Node*	<i>Rope</i> yang akan digunakan
2	x	Integer	Posisi indeks awal yang akan displit
3	len	Integer	Panjang <i>string</i> yang akan dipotong
4	fir	Pair Node*	Penyimpan hasil <i>split</i> pertama
5	sec	Pair Node*	Penyimpan hasil <i>split</i> kedua

3.1.9 Desain Fungsi Mutable End

Fungsi Mutable End digunakan untuk menjawab pertanyaan dengan *type* = 2. Hal ini bertujuan untuk menggabungkan hasil potongan *rope* pada bagian belakang *rope*. Fungsi ini memanfaatkan dua buah operasi *split* dan dua buah operasi *concat*. *Pseudocode* Fungsi Mutable End ditunjukkan pada Gambar 3.10.

Masukan	<i>rope</i> , posisi indeks, panjang <i>string</i>
Keluaran	Fungsi <code>concat(concat(<i>fir.first</i>, <i>sec.second</i>), <i>sec.first</i>)</code>
Print(<i>r</i> , <i>pos</i>) 1 <i>fir</i> = <code>split(<i>x</i>, <i>pos</i>)</code> 2 <i>sec</i> = <code>split(<i>fir.second</i>, <i>len</i>)</code> 3 return <code>concat(concat(<i>fir.first</i>, <i>sec.second</i>), <i>sec.first</i>)</code>	

Gambar 3.10 Pseudocode Fungsi Mutable End

Daftar variabel yang digunakan pada fungsi Mutable End dapat dilihat pada Tabel 3.9.

Tabel 3.9 Daftar Variable Untuk Fungsi Mutable End

No	Nama Variable	Tipe Data	Keterangan
1	<i>r</i>	Node*	<i>Rope</i> yang akan digunakan
2	<i>x</i>	Integer	Posisi indeks awal yang akan <i>displit</i>
3	<i>len</i>	Integer	Panjang <i>string</i> yang akan dipotong
4	<i>fir</i>	Pair Node*	Penyimpan hasil <i>split</i> pertama
5	<i>sec</i>	Pair Node*	Penyimpan hasil <i>split</i> kedua

Halaman ini sengaja dikosongkan

BAB IV

IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi dari desain dan algoritma penyelesaian permasalahan klasik *Alphabetic Rope*.

4.1 Lingkungan Implementasi

Lingkungan implementasi dalam pembuatan Tugas Akhir ini meliputi perangkat keras dan perangkat lunak yang digunakan untuk penyelesaian permasalahan klasik *Alphabetic Rope* adalah sebagai berikut:

1. Perangkat Keras:
 - *Processor* Intel(R) Core(TM) i3 - M 330 CPU @ 2.13GHz x 4
 - Memori 4 GB.
2. Perangkat Lunak:
 - Sistem operasi Ubuntu 14.04 LTS 64 bit.
 - *Text editor* Sublime Text 3.
 - *Compiler* g++ versi 4.3.2.

4.2 Rancangan Data

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma, dan data keluaran yang dihasilkan oleh program.

4.2.1 Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan menggunakan algoritma dan struktur data yang telah dirancang dalam Tugas Akhir ini.

Data masukan berupa berkas teks yang berisi data dengan format yang telah ditentukan pada deskripsi permasalahan klasik *Alphabetic Rope*. Pada masing-masing berkas data masukan, baris pertama berupa sebuah bilangan bulat yang merepresentasikan jumlah kasus uji yang ada pada berkas tersebut. Untuk setiap kasus uji dengan tipe 1 dan 2, masukan berupa sebuah baris masukan yang terdiri dari dua buah parameter posisi berupa x dan y . Sedangkan pada kasus uji dengan tipe 3, masukan berupa sebuah parameter y yang merupakan indeks *string* yang dicari pada konfigurasi *rope* saat ini.

4.2.2 Data Keluaran

Data keluaran yang dihasilkan oleh program hanya berupa satu nilai, yaitu huruf pada indeks ke- y untuk setiap kasus uji dengan tipe 3.

4.3 Implementasi Algoritma

Pada subbab ini akan dijelaskan tentang implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada Bab III.

4.3.1 Header yang Diperlukan

Implementasi algoritma dengan pemanfaatan struktur data Rope untuk menyelesaikan permasalahan klasik *Alphabetic Rope* membutuhkan empat buah *header* yaitu `stdio.h`, `cstdlib`, `cstring` dan `utility`, seperti yang terlihat pada Kode Sumber 4.1.

```

1  #include <stdio.h>
2  #include <cstdlib>
3  #include <cstring>
4  #include <utility>

```

Kode Sumber 4.1 *Header* yang diperlukan

Header `stdio.h` berisi modul untuk menerima masukan dan memberikan keluaran. *Header* `cstdlib` berisi modul untuk manajemen memori dinamis, generasi bilangan acak, pemilahan dan konversi. *Header* `cstring` berisi modul yang memiliki fungsi-fungsi untuk melakukan pemrosesan *string*. *Header* `utility` mencakup berbagai modul yang menyediakan fungsionalitas mulai dari aplikasi perhitungan bit hingga aplikasi fungsi parsial. Contoh implementasinya penggunaan *pair*.

4.3.2 Variabel Global

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan lintas fungsi. Kode sumber implementasi variabel global dapat dilihat pada Kode Sumber 4.2.

```

1  using namespace std;
2
3  const int N = 1e5 + 10;

```

Kode Sumber 4.2 Variabel Global

4.3.3 Implementasi Fungsi Main

Fungsi Main adalah implementasi algoritma yang dirancang pada Gambar 3.1. Setiap tipe memiliki operasi yang berbeda-beda. Untuk operasi dengan tipe 3, masukan hanya berupa parameter y yang merupakan nilai posisi indeks yang dicari pada konfigurasi

rope saat ini. Pada operasi dengan tipe 1 dan 2, baris masukan berupa nilai x dan y yang merupakan posisi indeks dari *string* pada *rope*. Setiap masukan dengan tipe 3 akan ditampilkan sebagai jawaban akhir dari permasalahan. Implementasi fungsi Main dapat dilihat pada Kode Sumber 4.3.

```

1  int main() {
2      char st[N];
3      scanf("%s", st);
4      Node* root = 0;
5      int Q, type;
6      root = insert(root, st);
7      scanf("%d", &Q);
8      while ( Q-- ) {
9          int x, y;
10         scanf("%d", &type);
11         if (type == 3) {
12             scanf("%d", &y);
13             printf("%c\n", print(root, y));
14         } else {
15             scanf("%d%d", &x, &y);
16             if (type == 1) {
17                 root = mutable_begin(root, x, y - x + 1);
18             } else {
19                 root = mutable_end(root, x, y - x + 1);
20             }
21         }
22     }
23     return 0;
24 }
```

Kode Sumber 4.3 Fungsi Main

4.3.4 Implementasi Struct Node

Fungsi Struct Node berisi atribut yang dimiliki *node* pada *tree*. Implementasi dari fungsi Struct Node dapat dilihat pada Kode Sumber 4.4.


```

1  struct Node {
2      Node *left, *right;
3      int size;
4      char value;
5      Node(char v) {
6          left = right = 0;
7          size = 1;
8          value = v;
9      }
10
11     Node* update() {
12         size = 1;
13         if ( left ) size += left->size;
14         if ( right ) size += right->size;
15         return this;
16     }
17 };

```

Kode Sumber 4.4 Fungsi Struct Node

4.3.5 Implementasi Fungsi Newnode

Fungsi Newnode digunakan untuk membentuk sebuah *node* yang berisikan karakter yang diberikan dan relasi terhadap karakter pada posisi yang bersebelahan. Sehingga membentuk sebuah *tree* seperti pada Gambar 2.4. Implementasi fungsi newnode dapat dilihat pada Kode Sumber 4.5.

```

1  Node* newnode(char c, Node* left, Node* right) {
2      Node* r = new Node(c);
3      r->left = left;
4      r->right = right;
5      r->update();
6      return r;
7  }

```

Kode Sumber 4.5 Fungsi Newnode

4.3.6 Implementasi Fungsi Build

Fungsi Build membangun struktur *tree* dari *rope* yang dilakukan dari karakter yang berada pada posisi indeks di tengah sampai pada karakter paling awal maupun akhir. Setiap *node* akan memiliki anak kiri dan anak kanan jika dan hanya jika masih terdapat *string* yang tersisa. Ilustrasinya dapat dilihat pada Gambar 2.5. Implementasi dari Fungsi Build dapat dilihat pada Kode Sumber 4.6.

```

1 Node* build(char* start, char* end) {
2     if ( start == end ) return NULL;
3     char* mid = start + (end - start)/2;
4     return newnode(*mid, build(start, mid),\
5         build(mid+1, end));
6 }
```

Kode Sumber 4.6 Fungsi Build

4.3.7 Implementasi Fungsi Getsize

Fungsi Getsize digunakan untuk mendapatkan berat *node* yang diperlukan. Implementasi dari fungsi Getsize dapat dilihat pada Kode Sumber 4.7.

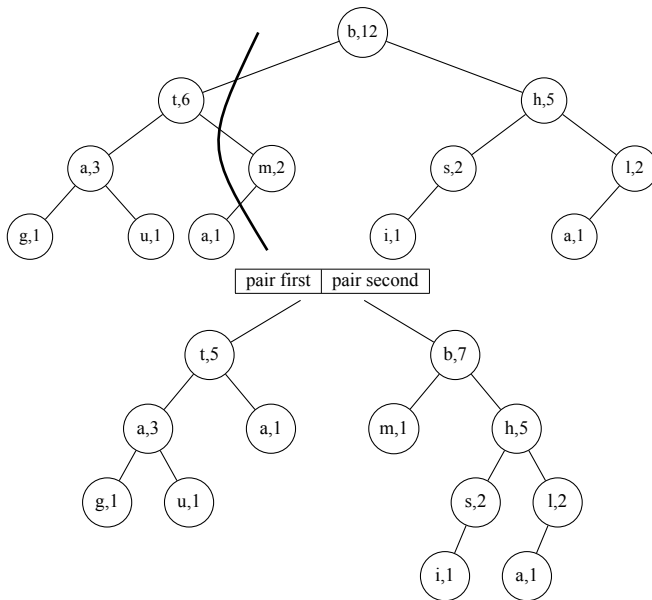
```

1 int getSize(Node* o) {
2     return o ? o->size : 0;
3 }
```

Kode Sumber 4.7 Fungsi Getsize

4.3.8 Implementasi Fungsi Split

Fungsi Split memanfaatkan struktur data Pair yang berupa pasangan dari dua *pointer* menuju *node*. *Pointer node* pertama akan berisi semua *node* yang bernilai lebih kecil dari posisi indeks parameter *split*, sedangkan *pointer node* kedua berisi semua *node* yang bernilai



Gambar 4.1 Ilustrasi Penyimpanan Hasil Operasi *Split Rope* pada Struktur Data Pair

lebih besar sama dengan posisi indeks parameter *split*.

Misalkan dilakukan *split* pada indeks ke-5 dari *rope* pada Gambar 4.1, *pointer node* pertama akan menyimpan semua *node* dari indeks ke-0 sampai dengan 4. Sedangkan *pointer node* kedua menyimpan *node* dari indeks ke-5 sampai akhir. Implementasi dari fungsi *split* dapat dilihat pada Kode Sumber 4.8 dan 4.9.

```

1 pair<Node*, Node*> split(Node* r, int pos) {
2     Node *R1 = 0, *R2 = 0;
3     if ( !r ) return make_pair(R1, R2);
4     int idx = getSize(r->left);
5     if ( idx < pos ) {

```

Kode Sumber 4.8 Fungsi Split(1)

```

1      pair<Node*, Node*> temp =\
2          split(r->right, pos - idx - 1);
3      r->right = temp.first;
4      R2 = temp.second;
5      R1 = r;
6  } else {
7      pair<Node*, Node*> temp = split(r->left, pos);
8      R1 = temp.first;
9      r->left = temp.second;
10     R2 = r;
11 }
12 r->update();
13 return make_pair(R1, R2);
14 }
```

Kode Sumber 4.9 Fungsi Split(2)

4.3.9 Implementasi Fungsi Random

Fungsi Random digunakan untuk menentukan posisi *node* pada saat penggabungan dua buah *rope*. Digunakan fungsi Random agar posisi *node* tidak konstan dan berat suatu *rope* tidak selalu sama. Implementasi fungsi Random ditunjukkan pada Kode Sumber 4.10.

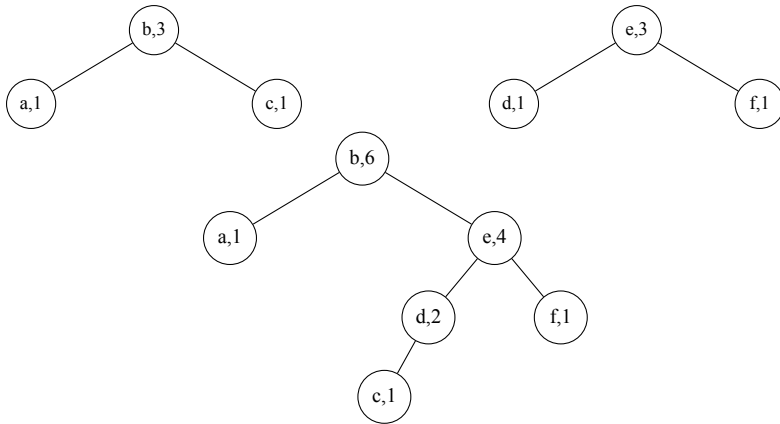
```

1 bool random(int a, int b) {
2     return rand() % ( a + b ) < a;
3 }
```

Kode Sumber 4.10 Fungsi Random

4.3.10 Implementasi Fungsi Concat

Fungsi Concat menggabungkan dua buah *rope* menjadi sebuah *rope* utuh. Untuk setiap *node* yang memiliki hasil modular nilai acak lebih kecil dari berat *node root rope* pertama, akan menjadi



Gambar 4.2 Ilustrasi Operasi Concat. Setiap *Node* Berisi Sebuah Karakter dan Nilai Berat Masing-Masing *Node*

root dari *rope* yang baru terbentuk. Untuk *rope* kedua akan menjadi anak kanan atau anak kirinya, tergantung pada urutan BST yang seharusnya dibuat.

Misalkan terdapat dua buah *rope* R_1 yang berisi *string* "abc" dan R_2 yang berisi *string* "def" seperti pada Gambar 4.2. Apabila akan digabungkan dengan R_1 berada di posisi depan sehingga membentuk *string* "abcdef", maka semua *node* pada R_2 berada di posisi kanan *rope* R_1 . Jika nilai hasil modular R_1 lebih kecil dari berat *node* *root* R_1 , *node* tersebut akan menjadi *root* dari *rope* yang baru terbentuk. *Node* *root* R_2 akan menjadi anak kanannya. Jika *root* R_1 memiliki anak kanan, akan menjadi anak kiri dari *node* R_1 yang memiliki indeks 0. Implementasi Fungsi Concat ditunjukkan pada Kode Sumber 4.11 dan 4.12.

```

1 Node* concat(Node* R1, Node* R2) {
2     if ( !R1 || !R2) return R1 ? R1 : R2;

```

Kode Sumber 4.11 Fungsi Concat(1)

```

1  if (random(R1->size, R2->size)) {
2      R1->right = concat(R1->right, R2);
3      return R1->update();
4  } else {
5      R2->left = concat(R1, R2->left);
6      return R2->update();
7  }
8  }

```

Kode Sumber 4.12 Fungsi Concat(2)

4.3.11 Implementasi Fungsi Insert

Fungsi Insert adalah implementasi dari desain algoritma pada Gambar 3.2. Fungsi ini bertujuan untuk memasukkan data *string* ke dalam *rope*. Setiap *string* akan dimasukkan ke dalam suatu *node*. Setiap *node* hanya berisi oleh satu karakter pecahan dari *string* masukan. Implementasi dari Fungsi Insert dapat dilihat pada Kode Sumber 4.13.

```

1  Node* insert(Node* r, char s[]) {
2      Node* x = build(s, s + strlen(s));
3      return concat(r, x);
4  }

```

Kode Sumber 4.13 Fungsi Insert

4.3.12 Implementasi Fungsi Mutable Begin

Fungsi Mutable Begin adalah implementasi dari desain algoritma pada Gambar 3.9. Operasi ini dilakukan untuk menjawab permasalahan pada subbab 2.1.1 dengan memanfaatkan dua buah operasi Split dan dua buah operasi Concat.

Misal dilakukan operasi 1 5 9 pada *rope* di Gambar 4.3. Maka *rope* akan dipotong pada posisi indeks ke-5 menghasilkan *rope* R_1 dan

R_2 . Kemudian dilakukan *split* kedua kali pada indeks $y - x + 1$ pada *rope* R_2 menghasilkan *rope* R_{21} dan R_{22} . Sehingga menghasilkan tiga buah *rope* yang disimpan dalam struktur data Pair. Langkah selanjutnya dilakukan operasi Concat pada *rope* R_1 dengan R_{22} . Hasil penggabungan *rope* R_1 dengan R_{22} akan digabungkan dengan *rope* R_{21} . Pada operasi ini, *rope* R_{21} akan berada di posisi paling kiri dari keseluruhan *rope*. Dan menghasilkan sebuah *rope* utuh dengan urutan posisi *rope* saat ini adalah R_{21} , R_1 dan R_{22} . Implementasi Fungsi Mutable Begin dapat dilihat pada Kode Sumber 4.14.

```

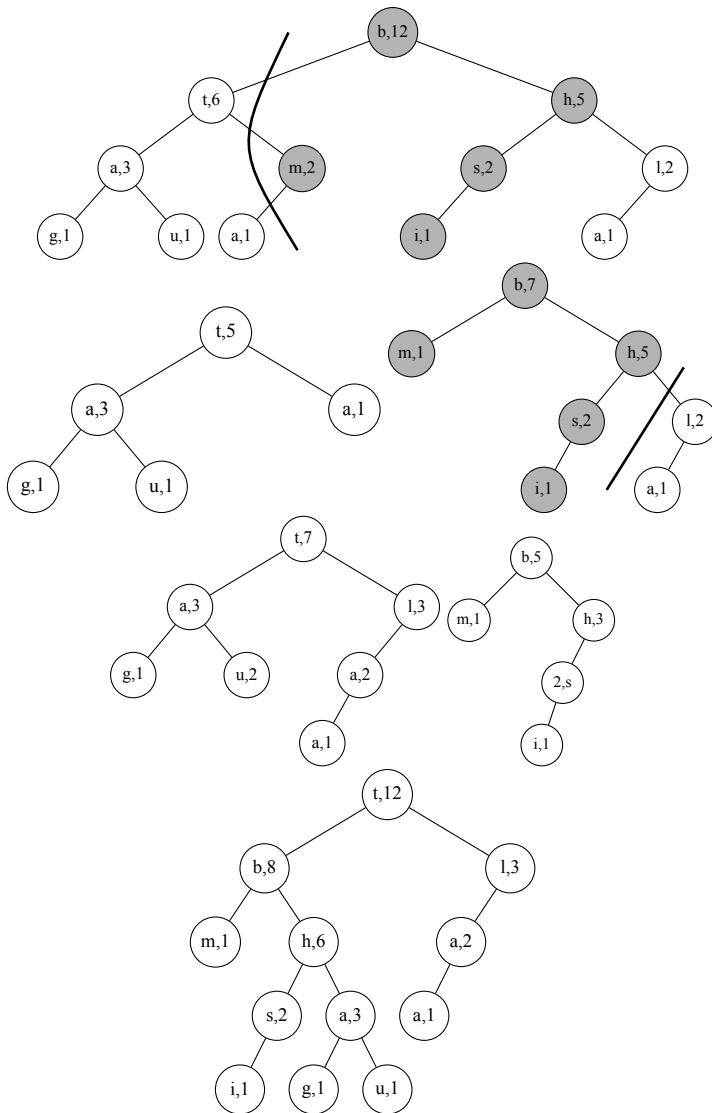
1 Node* mutable_begin(Node* r, int x, int len) {
2     pair<Node*, Node*> fir = split(r, x);
3     pair<Node*, Node*> sec = split(fir.second, len);
4     return concat(sec.first, concat(fir.first, \
5         sec.second));
6 }
```

Kode Sumber 4.14 Fungsi Mutable Begin

4.3.13 Implementasi Fungsi Mutable End

Fungsi Mutable End dilakukan untuk menjawab permasalahan pada subbab 2.1.2. Implementasinya dilakukan dengan memanfaatkan dua buah operasi Split dan dua buah operasi Concat.

Misal dilakukan operasi 2 5 9 pada *rope* di Gambar 4.4. Maka *rope* akan dipotong pada posisi indeks ke-5 menghasilkan *rope* R_1 dan R_2 . Kemudian dilakukan *split* kedua kali pada indeks $y - x + 1$ pada *rope* R_2 menghasilkan *rope* R_{21} dan R_{22} . Sehingga menghasilkan tiga buah *rope* yang disimpan dalam struktur data Pair. Langkah selanjutnya dilakukan operasi Concat pada *rope* R_1 dengan R_{22} . Hasil penggabungan *rope* R_1 dengan R_{22} akan digabungkan dengan *rope* R_{21} . Pada operasi ini, *rope* R_{21} akan berada di posisi paling kanan dari keseluruhan *rope*. Dan menghasilkan sebuah *rope* utuh dengan urutan posisi *rope* saat ini adalah R_1 , R_{22} dan R_{21} .



Gambar 4.3 Ilustrasi Operasi Mutable Begin. Setiap *Node* Berisi Sebuah Karakter dan Nilai Prioritas Masing-Masing *Node*

Implementasi Fungsi Mutable End dapat dilihat pada Kode Sumber 4.15.

```

1 Node* mutable_end(Node* r, int x, int len) {
2     pair<Node*, Node*> fir = split(r, x);
3     pair<Node*, Node*> sec = split(fir.second, len);
4     return concate(concate(fir.first, sec.second), \
5         sec.first);
6 }

```

Kode Sumber 4.15 Fungsi Mutable End

4.3.14 Implementasi Fungsi Print

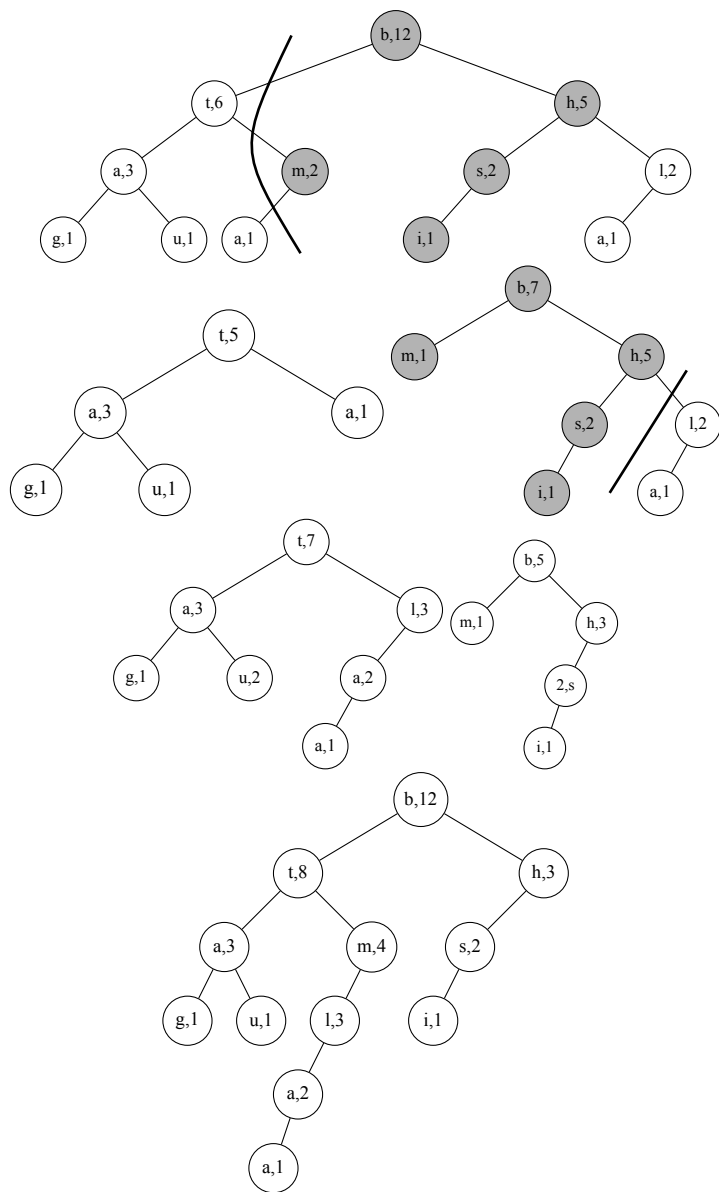
Fungsi Print digunakan untuk menjawab permasalahan klasik *Alphabetic Rope* sesuai dengan permasalahan pada subbab 2.1.3. Implementasi fungsi ini berdasarkan dari desain algoritma pada Gambar 3.8. Implementasi dari Fungsi Print dapat dilihat pada Kode Sumber 4.16.

```

1 char print(Node* r, int pos) {
2     int idx = getSize(r->left);
3     if ( idx == pos ) return r->value;
4     if ( pos <= idx ) return print(r->left, pos);
5     else return print(r->right, pos - idx - 1);
6 }

```

Kode Sumber 4.16 Fungsi Print



Gambar 4.4 Ilustrasi Operasi Mutable End. Setiap *Node* Berisi Sebuah Karakter dan Nilai Prioritas Masing-Masing *Node*

BAB V

UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada Tugas Akhir ini.

5.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan adalah salah satu sistem yang digunakan situs penilaian daring SPOJ, yaitu kluster *Cube* dengan spesifikasi sebagai berikut:

1. Perangkat Keras:
 - *Processor* Intel(R) Pentium G860 CPU @ 3GHz.
 - *Memory* 1536 MB.
2. Perangkat Lunak:
 - *Compiler* g++ versi 4.3.2.

5.2 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan analisis penyelesaian sebuah contoh kasus menggunakan pendekatan penyelesaian yang telah dijelaskan pada subbab 2.2.4 serta pengumpulan berkas kode sumber hasil implementasi ke dalam situs penilaian daring SPOJ.

Kasus yang akan digunakan sebagai bahan uji kebenaran dalam analisis penyelesaian permasalahan klasik *Alphabetic Rope* menggunakan contoh kasus pada Gambar 5.1.

Mula-mula dalam setiap permasalahan adalah membangun struktur *rope* yang sesuai dengan *string* masukan. Operasi dasar pada sebuah *rope* adalah fungsi Insert, dimana dibentuk sebuah *tree* yang berfungsi untuk menyimpan potongan karakter pada *rope*.

gautambishal
3
3 1
2 0 5
3 1

Gambar 5.1 Contoh kasus uji permasalahan Alphabetic Rope

Selanjutnya *rope* akan dibangun pada *node root* yang dilakukan berdasarkan posisi tengah dari panjang *string*. Pembentukan ini dapat dilihat pada Gambar 5.2. Nilai pada tanda kurung siku menyatakan isi karakter pada *node* tersebut. Dikarenakan masih terdapat *string* yang tersisa maka proses pembentukan dilanjutkan untuk membentuk anak kiri dan anak kanan *root*. Rentang *string*

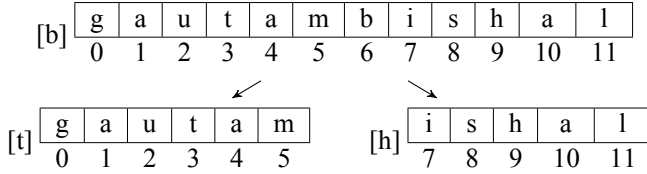
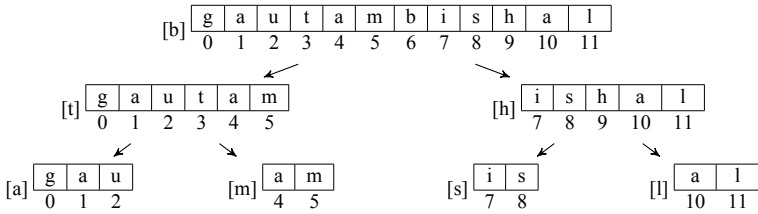
[b]	g	a	u	t	a	m	b	i	s	h	a	l
	0	1	2	3	4	5	6	7	8	9	10	11

Gambar 5.2 Pembentukan *Root Rope*

pada anak kiri diperoleh dari indeks 0 hingga nilai tengah-1 sedangkan rentang kanan diperoleh dari nilai tengah+1 hingga panjang *string*. Pembentukan anak dari *root* ditunjukkan pada Gambar 5.3. Pada setiap anak kiri dan anak kanan, posisi karakter yang berada ditengah-tengah *string* anak menjadi nilai dari *node* tersebut. Yang ditunjukkan oleh nilai di dalam tanda kurung siku. Dikarenakan masih terdapat *string* yang tersisa, proses pembentukan dilanjutkan untuk tingkat ke-3 yang ditunjukkan pada Gambar 5.4.

Untuk tahap selanjutnya dilakukan pembentukan *rope* pada tingkat ke-4 yang ditunjukkan pada Gambar 5.5.

Query pertama memiliki parameter *type* = 3, *y* = 0. Tahapan pertama dengan mencari karakter pada indeks ke-*y*. Untuk

Gambar 5.3 Pembentukan *Rope* pada Tingkat ke-2Gambar 5.4 Pembentukan *Rope* pada Tingkat ke-3

memperoleh jawaban dilakukan penelusuran sesuai dengan algoritma yang dijelaskan pada subbab 2.2.4.3.

Tahapan pencarian pada *rope* adalah sebagai berikut,

$y = 1$, $left.weight = 6$, $idx = left.weight$; $idx \leq y$;
dilanjutkan ke anak kiri;

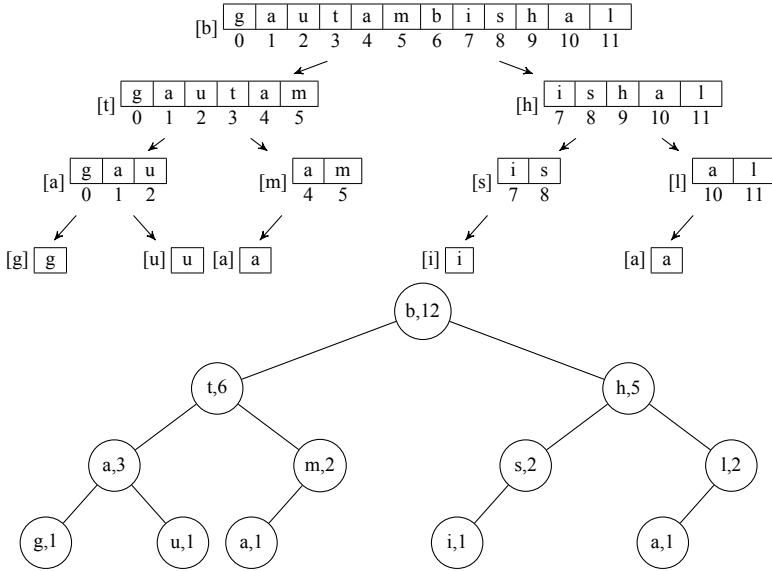
$y = 1$, $left.weight = 3$, $idx = left.weight$; $idx \leq y$;
dilanjutkan ke anak kiri;

$y = 1$, $left.weight = 1$, $idx = left.weight$; $idx = y$;
kembalikan nilai *node* saat ini;

Maka jawaban untuk *query* ini adalah *a*.

Query kedua memiliki parameter $type = 2$, $x = 0$, $y = 5$. Proses perubahan pada *rope* adalah sebagai berikut,

$x = 0$, $y = 5$, $left.weight = 6$, $idx = left.weight$; $idx \leq y$;

Gambar 5.5 Struktur *Rope* yang Terbentuk

dilanjutkan ke anak kiri;

$x = 0, y = 5, left.weight = 3, idx = left.weight; idx \leq y;$
 dilanjutkan ke anak kiri;

$x = 0, y = 5, left.weight = 1, idx = left.weight; idx \leq y;$
 dilanjutkan ke anak kiri;

$x = 0, y = 5, left.weight = 0, idx = left.weight; idx = y;$
 kembalikan nilai *node* saat ini;

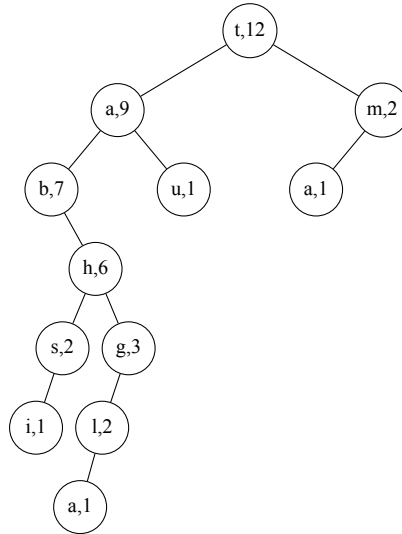
Dikarenakan $x = 0$ operasi Split menghasilkan *rope* R_1 yang berisi NULL dan R_2 sisa dari potongan *rope* yang berarti keseluruhan *rope* saat ini.

Proses perubahan selanjutnya adalah sebagai berikut,

$x = 0, y = 5, left.weight = 1, len = y - x + 1, idx = left.weight; idx = len;$ kembalikan nilai *node* saat ini;

Setelah menemukan posisi *node* yang akan dipotong, hapus sambungan dari seluruh *node* yang memiliki nilai indeks kurang dari nilai indeks saat ini. Hasil potongan dari *rope* akan disimpan dengan menggunakan struktur data Pair yang bertipe *pointer node*. Sehingga menghasilkan dua buah *rope* baru R_{21} yang berisi string "gautam" dan R_{22} yang berisi string "bishal".

Potongan *rope* yang baru terbentuk akan digabungkan berdasarkan $type = 2$, dimana R_{21} berada diposisi belakang dari keseluruhan *rope*. Konfigurasi *rope* yang baru terbentuk berisi string "bishalgautam" yang ditunjukkan pada Gambar 5.6.



Gambar 5.6 Konfigurasi *Rope* Setelah Operasi 2 0 5

Query ketiga memiliki parameter $type = 3, y = 0$. Proses perubahan dan pencarian pada *rope* adalah sebagai berikut,

$y = 0$, $left.weight = 9$, $idx = left.weight$; $idx \leq y$;
dilanjutkan ke anak kiri;

$y = 0$, $left.weight = 7$, $idx = left.weight$; $idx \leq y$;
dilanjutkan ke anak kiri;

$y = 0$, $left.weight = 0$, $idx = left.weight$; $idx = y$;
kembalikan nilai *node* saat ini;

Maka jawaban untuk *query* ini adalah *b*.

Secara terurut jawaban dari kedua *query* tersebut adalah *a* dan *b*. Kemudian sistem penyelesaian dijalankan dan diberi masukan sesuai kasus uji dari analisis sebelumnya dan hasil luaran sistem adalah *a* dan *b* seperti terlihat pada Gambar 5.7.

```
gautambishal
3
3 1
a
2 0 5
3 0
b
```

Gambar 5.7 Hasil Luaran Program pada Contoh Kasus Uji Alphabetic Rope

Selanjutnya dilakukan juga uji coba kebenaran dengan mengirimkan kode sumber program ke dalam situs penilaian daring SPOJ. Permasalahan yang diselesaikan adalah permasalahan klasik *Alphabetic Rope*. Hasil uji kebenaran dan waktu eksekusi program pada saat pengumpulan kasus uji pada situs SPOJ ditunjukkan pada Gambar 5.8.

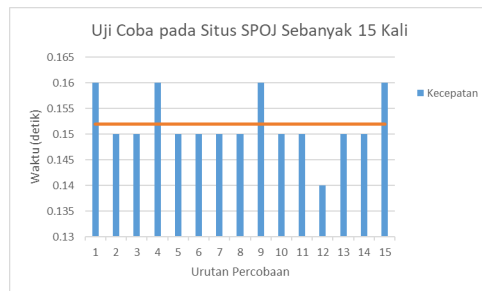
20966212	2018-01-15 16:18:03	Alphabetic Rope	accepted edit ideone.it	0.15	5.0M	C++ 4.3.2
----------	------------------------	-----------------	----------------------------	------	------	--------------

Gambar 5.8 Hasil Uji Coba pada Situs Penilaian SPOJ

Tabel 5.1 Kecepatan Maksimal, Minimal dan Rata-Rata dari Hasil Uji Coba Sebanyak 15 Kali pada Situs Pengujian SPOJ

Waktu Maksimal	0.16 detik
Waktu Minimal	0.14 detik
Waktu Rata-Rata	0.152 detik
Memori Maksimal	5.0 MB
Memori Minimal	5.0 MB
Memori Rata-rata	5.0 MB

Hal ini membuktikan bahwa implementasi yang dilakukan telah berhasil menyelesaikan permasalahan klasik *Alphabetic Rope* dengan batasan-batasan yang telah ditetapkan. Setelah itu dilakukan pengiriman kode sumber implementasi sebanyak 15 kali untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 15 kali dapat dilihat pada Gambar A.1. Grafik hasil uji coba sebanyak 15 kali ditunjukkan pada Gambar 5.9.



Gambar 5.9 Hasil Uji Coba pada Situs Penilaian SPOJ

Berdasarkan Tabel 5.1 dari percobaan pengujian yang dilakukan, didapat waktu rata-rata program yaitu 0.152 detik dan penggunaan memori yang dibutuhkan program yaitu 5.0 MB. Hasil ini masih

dibawah dari batas maksimal waktu dan memori pada permasalahan klasik *Alphabetic Rope*, yaitu 1 detik dan 1536 MB.

5.3 Uji Coba Kinerja

Uji coba kinerja penyelesaian permasalahan klasik *Alphabetic Rope* dilakukan dengan cara membandingkan waktu yang dibutuhkan untuk penyelesaian menggunakan *string* dengan struktur data Rope yang dibuat.

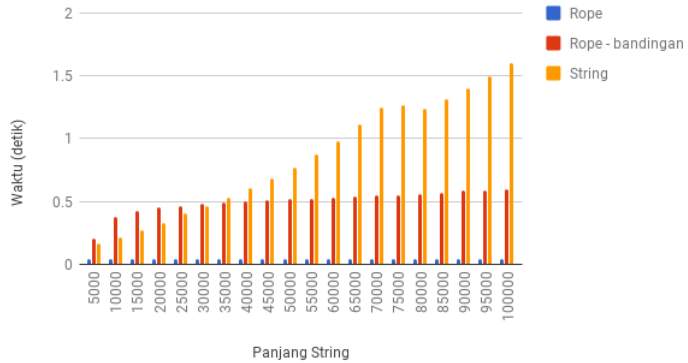
5.3.1 Operasi 1 Menggabungkan *Rope* pada Posisi Awal

Pada Gambar 5.10 terlihat bahwa untuk $N = 5000$ sampai $N = 10^5$ dengan banyak $Q = 10^5$, struktur Rope yang dibuat untuk penyelesaian permasalahan klasik *Alphabetic Rope* menunjukkan waktu yang lebih cepat dibandingkan dengan algoritma String dan stuktur data Rope pembanding. Sedangkan untuk $N = 10^5$ dengan $Q = 5000$ sampai $Q = 10^5$, pertumbuhan waktu secara logaritmik dan lebih kecil dibandingkan dengan algoritma String dan struktur data Rope pembanding yang ditunjukkan pada Gambar 5.11. Untuk tabel performa antara algoritma String dengan struktur data Rope yang telah dibuat dapat dilihat pada Tabel B.1 dan B.2.

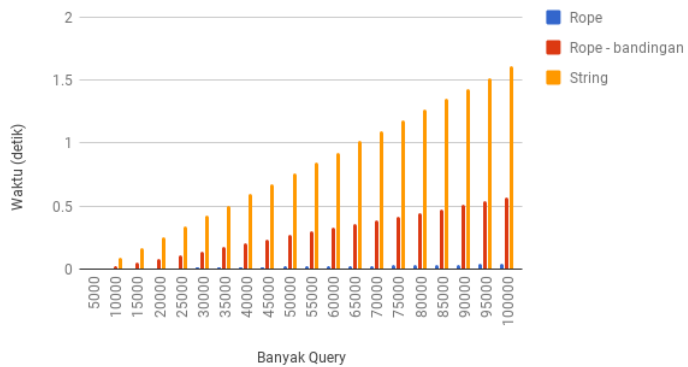
5.3.2 Operasi 2 Menggabungkan *Rope* pada Posisi Akhir

Pada Gambar 5.12 terlihat bahwa untuk $N = 5000$ sampai $N = 10^5$ dengan banyak $Q = 10^5$, struktur Rope yang dibuat untuk penyelesaian permasalahan klasik *Alphabetic Rope* menunjukkan waktu yang lebih cepat dibandingkan dengan algoritma String dan stuktur data Rope pembanding. Sedangkan untuk $N = 10^5$ dengan $Q = 5000$ sampai $Q = 10^5$, pertumbuhan waktu secara logaritmik dan lebih kecil dibandingkan dengan algoritma String dan struktur data Rope pembanding yang ditunjukkan pada Gambar 5.13. Untuk

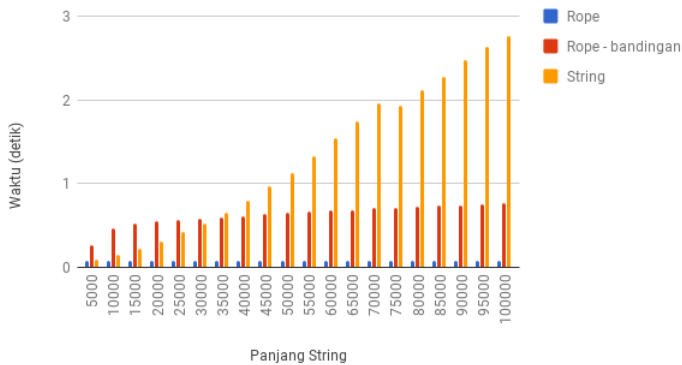
Perbandingan Performa Operasi 1

Gambar 5.10 Hasil Uji Coba pada Operasi 1 dengan Jumlah *Query* Tetap dan Panjang *String* Bertambah

Perbandingan performa Operasi 1

Gambar 5.11 Hasil Uji Coba pada Operasi 1 dengan Jumlah *String* Tetap dan Jumlah *Query* Bertambah

Perbandingan Performa Operasi 2

Gambar 5.12 Hasil Uji Coba pada Operasi 2 dengan Jumlah *Query* Tetap dan Jumlah *String* Bertambah

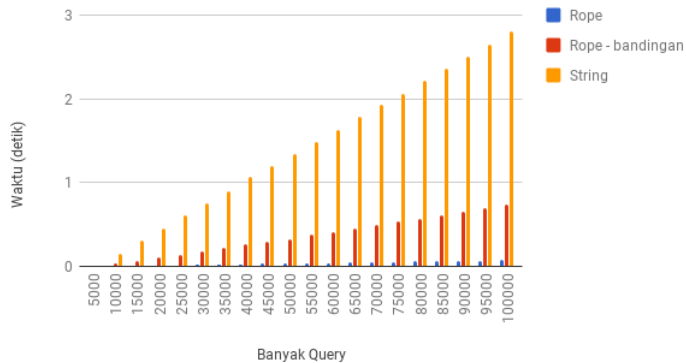
tabel performa antara algoritma String dengan struktur data Rope yang telah dibuat dapat dilihat pada Tabel C.1 dan C.2.

5.3.3 Operasi 3 Mencetak Karakter pada Indeks ke-Y

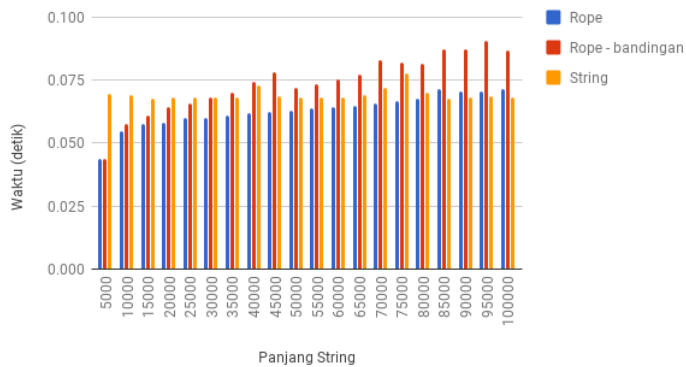
Pada Gambar 5.14 terlihat bahwa untuk $N = 5000$ sampai $N = 10^5$ dengan banyak $Q = 10^5$, algoritma String untuk penyelesaian permasalahan klasik *Alphabetic Rope* menunjukkan waktu yang lebih cepat dibandingkan dengan struktur data Rope yang dibuat dan stuktur data Rope pembanding. Dan berlaku ketika menjalankan operasi untuk $N = 10^5$ dengan $Q = 5000$ sampai $Q = 10^5$ yang ditunjukkan pada Gambar 5.15. Untuk tabel performa antara algoritma String dengan struktur data Rope yang telah dibuat dapat dilihat pada Tabel D.1 dan D.2.

Perbedaan *rope* dengan *rope*-bandingan terletak pada operasi *insert*. Pada *rope* bandingan operasi *insert* dilakukan sama seperti algoritma *insert* pada BST. Sedangkan pada *rope*, *insert* dilakukan

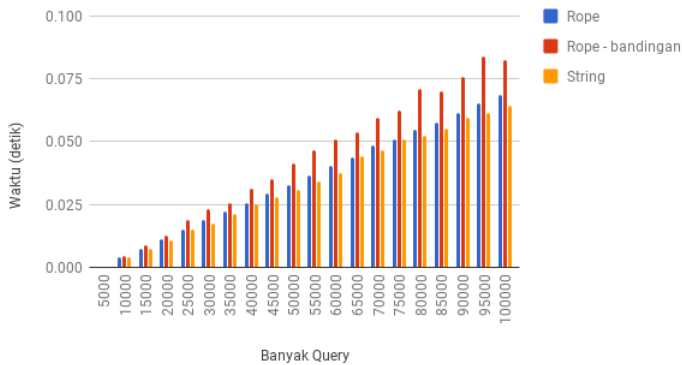
Perbandingan Performa Operasi 2

Gambar 5.13 Hasil Uji Coba pada Operasi 2 dengan Panjang *String* Tetap dan Jumlah *Query* Bertambah

Perbandingan Performa Operasi 3

Gambar 5.14 Hasil Uji Coba pada Operasi 3 dengan Jumlah *Query* Tetap dan Panjang *String* Bertambah

Perbandingan Performa Operasi 3



Gambar 5.15 Hasil Uji Coba pada Operasi 3 dengan Jumlah *String* Tetap dan Jumlah *Query* Bertambah

dengan membagi dua panjang *string* kemudian dimasukkan ke dalam *tree*.

5.4 Analisis Hasil Uji Coba

Pada subbab ini akan dibahas mengenai analisis hasil uji coba yang dikerjakan pada subbab 5.2. Berdasarkan ketiga skenario yang telah dilakukan, masing-masing membuktikan kebenaran dan efisiensi hasil implementasi Tugas Akhir ini.

Dari ketiga hasil uji coba yang telah dilakukan, dapat dilihat bahwa hasil implementasi algoritma penyelesaian permasalahan klasik *Alphabetic Rope* pada Tugas Akhir ini mengeluarkan hasil keluaran yang sama dengan jalur yang ditelusuri secara manual. Hasil uji coba kebenaran pada subbab 5.2 dapat digunakan sebagai acuan bahwa hasil keluaran program akan benar untuk segala jenis operasi pertanyaan dan perubahan.

Selanjutnya berdasarkan uji coba performa yang ditunjukkan

pada Gambar A.1, dapat dilihat bahwa memori dan waktu yang dibutuhkan untuk mengeksekusi program dapat dikategorikan efisien menurut situs SPOJ.

Proses *split*, *concat* dan *print* pada *rope* mengacu pada algoritma pencarian pada *tree* sehingga dapat disimpulkan bahwa kompleksitas waktu yang diperlukan sebesar $O(\log N)$. Pada proses pembuatan *rope* berdasarkan Kode Sumber 4.6 didapatkan kompleksitas waktu sebesar $O(N \log N)$. Sehingga dapat disimpulkan bahwa kompleksitas waktu untuk algoritma komputasi *string* pada Tugas Akhir ini adalah $O(\log N)$, dimana N merupakan panjang *string*. Sedangkan algoritma String membutuhkan kompleksitas sebesar $O(N)$ untuk setiap proses *split* dan *concat*[2].

Pada hasil perbandingan antara algoritma String dengan struktur Rope yang digunakan pada Tugas Akhir ini, ditunjukkan perbedaan pada kecepatan eksekusi program. Hal ini menunjukkan bahwa algoritma pada Tugas Akhir ini lebih efisien daripada algoritma String dan Rope-bandingan.

Halaman ini sengaja dikosongkan

BAB VI

KESIMPULAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap perancangan dan implementasi algoritma untuk menyelesaikan permasalahan klasik *Alphabetic Rope* dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma dengan menggunakan struktur data Rope dapat menyelesaikan permasalahan klasik *Alphabetic Rope* dengan benar.
2. Kompleksitas waktu sebesar $O(\log N)$ dapat menyelesaikan permasalahan klasik *Alphabetic Rope*.
3. Waktu yang dibutuhkan program untuk menyelesaikan permasalahan klasik *Alphabetic Rope* minimum 0.14 detik, maksimum 0.16 detik dan rata-rata 0.152 detik. Memori yang dibutuhkan sebesar 5.0 MB.
4. Struktur data Rope yang dibuat sangat baik diaplikasikan untuk melakukan komputasi *string* yang panjang.

6.2 Saran

Pada Tugas Akhir kali ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui Tugas Akhir ini:

1. Struktur data Rope adalah pendekatan yang sesuai untuk menyelesaikan permasalahan *string* yang sangat panjang.

2. Struktur data Rope dapat diimplementasikan pada bahasa pemrograman lain.

DAFTAR PUSTAKA

- [1] **AROEPE** - Alphabetic Rope [Online]. Available:
<http://www.spoj.com/problems/AROEPE/>. [Accessed
24-October-2017].
- [2] **Advance Programming with C++**. Available:
<http://www.acsu.buffalo.edu/fineberg/mfc158/week10lecture.htm>.
[Accessed 24-October-2017].
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, **Introduction To Algorithm**, 2nd ed. Cambridge, Massachusetts London, England: The MIT Press, 2001.
- [4] Cecilia R. Aragon, **Randomize Search Trees**, USA: Foundation of Computer Science, 1989.
- [5] Hans-J. Boehm, Russ Atkinson and Michael Pass, **Ropes: an Alternative to String**, California: Xerox PARC, 1994.

Halaman ini sengaja dikosongkan

LAMPIRAN A

HASIL UJI COBA KEBENARAN PADA SITUS SPOJ

20966263	<input type="checkbox"/>	2019-01-11 16:22:63	Alphabetic Rope	accepted edit delete it	0.16	5.0M	C++ 4.3.2
20966258	<input type="checkbox"/>	2019-01-11 16:21:46	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966253	<input type="checkbox"/>	2019-01-11 16:21:29	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966248	<input type="checkbox"/>	2019-01-11 16:21:08	Alphabetic Rope	accepted edit delete it	0.16	5.0M	C++ 4.3.2
20966241	<input type="checkbox"/>	2019-01-11 16:20:52	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966240	<input type="checkbox"/>	2019-01-11 16:20:35	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966238	<input type="checkbox"/>	2019-01-11 16:20:18	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966236	<input type="checkbox"/>	2019-01-11 16:20:01	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966231	<input type="checkbox"/>	2019-01-11 16:19:44	Alphabetic Rope	accepted edit delete it	0.16	5.0M	C++ 4.3.2
20966227	<input type="checkbox"/>	2019-01-11 16:19:26	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966224	<input type="checkbox"/>	2019-01-11 16:19:08	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966220	<input type="checkbox"/>	2019-01-11 16:18:45	Alphabetic Rope	accepted edit delete it	0.14	5.0M	C++ 4.3.2
20966218	<input type="checkbox"/>	2019-01-11 16:18:25	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966212	<input type="checkbox"/>	2019-01-11 16:18:03	Alphabetic Rope	accepted edit delete it	0.15	5.0M	C++ 4.3.2
20966209	<input type="checkbox"/>	2019-01-11 16:17:45	Alphabetic Rope	accepted edit delete it	0.16	5.0M	C++ 4.3.2

Gambar A.1 Hasil Pengujian Sebanyak 15 Kali pada Situs Penilaian Daring SPOJ

Halaman ini sengaja dikosongkan

LAMPIRAN B

**HASIL UJI PERFORMA PENYELESAIAN OPERASI 1
MENGUNAKAN STRUKTUR DATA ROPE DAN
ALGORITHM A STRING**

Tabel B.1 Tabel Hasil Uji Coba pada Operasi 1 dengan Jumlah *Query* tetap dan Panjang *String* Bertambah

Panjang String	Rope Waktu (detik)	Rope - pembandingan Waktu (detik)	String Waktu (detik)
5000	0.04584	0.37512	0.21461
10000	0.04568	0.42488	0.27259
15000	0.04298	0.45005	0.32707
20000	0.04261	0.46458	0.40244
25000	0.04352	0.47984	0.46670
30000	0.04239	0.48971	0.53264
35000	0.04229	0.49920	0.60791
40000	0.04324	0.50714	0.68570
45000	0.04716	0.51634	0.77109
50000	0.04239	0.52379	0.87648
55000	0.04225	0.53031	0.98290
60000	0.04218	0.53815	1.11034
65000	0.04262	0.54562	1.24909
70000	0.04251	0.55366	1.26256
75000	0.04286	0.56215	1.24042
80000	0.04524	0.57252	1.31375
85000	0.04271	0.58538	1.40470
90000	0.04243	0.58756	1.49502
95000	0.04246	0.60007	1.59853
100000	0.04229	0.60352	1.69388

Tabel B.2 Tabel Hasil Uji Coba pada Operasi 1 dengan Panjang *String* tetap dan Jumlah *Query* Bertambah

Banyak Query	Rope Waktu (detik)	Rope - pembandingan Waktu (detik)	String Waktu (detik)
5000	0.00212	0.02678	0.09447
10000	0.00426	0.05573	0.17078
15000	0.00634	0.08553	0.25521
20000	0.00851	0.11321	0.33912
25000	0.01048	0.14259	0.42499
30000	0.01272	0.17272	0.50647
35000	0.01482	0.20204	0.59519
40000	0.01777	0.23304	0.67729
45000	0.01924	0.27471	0.75871
50000	0.02168	0.29658	0.84541
55000	0.02483	0.32535	0.92717
60000	0.02616	0.3547	1.01493
65000	0.02785	0.38604	1.09944
70000	0.02959	0.41976	1.18202
75000	0.03220	0.44713	1.26605
80000	0.03432	0.47656	1.35644
85000	0.03614	0.50967	1.43311
90000	0.03923	0.53864	1.51598
95000	0.04044	0.56934	1.60935
100000	0.04229	0.60352	1.69388

LAMPIRAN C

**HASIL UJI PERFORMA PENYELESAIAN OPERASI 2
MENGUNAKAN STRUKTUR DATA ROPE DAN STRING**

Tabel C.1 Tabel Hasil Uji Coba pada Operasi 2 dengan Jumlah *Query* tetap dan Panjang *String* Bertambah

Panjang String	Rope Waktu (detik)	Rope - pembanding Waktu (detik)	String Waktu (detik)
5000	0.08046	0.45924	0.15061
10000	0.07907	0.51843	0.21968
15000	0.07804	0.54997	0.30699
20000	0.07941	0.56391	0.42457
25000	0.07919	0.58758	0.52275
30000	0.08137	0.59856	0.64959
35000	0.07963	0.6151	0.80127
40000	0.0807	0.6351	0.96337
45000	0.07853	0.64935	1.12678
50000	0.07917	0.66312	1.33048
55000	0.08274	0.67976	1.5374
60000	0.07855	0.68753	1.74437
65000	0.07848	0.70459	1.9582
70000	0.08031	0.71008	1.93652
75000	0.07816	0.73011	2.12236
80000	0.07947	0.73807	2.27391
85000	0.08009	0.7404	2.4679
90000	0.07838	0.74922	2.63696
95000	0.07819	0.76165	2.7606
100000	0.08379	0.76779	2.94229

Tabel C.2 Tabel Hasil Uji Coba pada Operasi 2 dengan Panjang String tetap dan Jumlah Query Bertambah

Banyak Query	Rope Waktu (detik)	Rope - pembanding Waktu (detik)	String Waktu (detik)
5000	0.0021	0.03094	0.15582
10000	0.00422	0.06923	0.31249
15000	0.0082	0.10347	0.45776
20000	0.01235	0.14144	0.60343
25000	0.01652	0.17569	0.75053
30000	0.02187	0.21542	0.89214
35000	0.02437	0.25814	1.06315
40000	0.02868	0.29264	1.19422
45000	0.03337	0.32757	1.33897
50000	0.03724	0.37407	1.48473
55000	0.04161	0.41413	1.62678
60000	0.04647	0.45708	1.77913
65000	0.04975	0.48762	1.92705
70000	0.05391	0.5315	2.06298
75000	0.05869	0.57248	2.21603
80000	0.06229	0.61257	2.35933
85000	0.06674	0.65026	2.50408
90000	0.07095	0.68927	2.64912
95000	0.07366	0.74459	2.80072
100000	0.08379	0.76779	2.94229

LAMPIRAN D

**HASIL UJI PERFORMA PENYELESAIAN OPERASI 3
MENGUNAKAN STRUKTUR DATA ROPE DAN STRING**

Tabel D.1 Tabel Hasil Uji Coba pada Operasi 3 dengan Jumlah *Query* tetap dan Panjang *String* Bertambah

Panjang String	Rope Waktu (detik)	Rope - pembanding Waktu (detik)	String Waktu (detik)
5000	0.05484	0.05753	0.06929
10000	0.05765	0.06085	0.06749
15000	0.05815	0.06444	0.06798
20000	0.06014	0.06593	0.06812
25000	0.06007	0.06808	0.06819
30000	0.06121	0.06983	0.06805
35000	0.06197	0.07425	0.07311
40000	0.06253	0.0781	0.06874
45000	0.06310	0.07199	0.06835
50000	0.06381	0.0733	0.06814
55000	0.06449	0.07532	0.06797
60000	0.06485	0.07707	0.0689
65000	0.06592	0.08287	0.07186
70000	0.06668	0.08182	0.07765
75000	0.06788	0.08158	0.06994
80000	0.07156	0.08723	0.06788
85000	0.07049	0.0875	0.06823
90000	0.07054	0.09047	0.06874
95000	0.07153	0.08684	0.06834
100000	0.07222	0.0871	0.07223

Tabel D.2 Tabel Hasil Uji Coba pada Operasi 3 dengan Panjang String tetap dan Jumlah Query Bertambah

Banyak Query	Rope Waktu (detik)	Rope - pembanding Waktu (detik)	String Waktu (detik)
5000	0.00403	0.0046	0.00411
10000	0.00753	0.00901	0.00729
15000	0.01116	0.01278	0.01078
20000	0.01524	0.01901	0.01501
25000	0.01878	0.02309	0.01763
30000	0.02204	0.02543	0.02134
35000	0.02565	0.03114	0.02496
40000	0.02920	0.03496	0.02786
45000	0.03271	0.04112	0.03096
50000	0.03638	0.04661	0.03399
55000	0.04029	0.05072	0.03738
60000	0.04361	0.05377	0.04412
65000	0.04844	0.05963	0.04669
70000	0.05078	0.06232	0.05085
75000	0.05493	0.07118	0.05242
80000	0.05782	0.06998	0.05502
85000	0.06153	0.0758	0.05955
90000	0.06514	0.08392	0.06156
95000	0.06884	0.08232	0.06455
100000	0.07222	0.0871	0.07223

BIODATA PENULIS



Desy Nurbaiti Rahmi, lahir di Banyuwangi tanggal 9 Desember 1995. Penulis merupakan anak ketiga dari 3 bersaudara. Penulis telah menempuh pendidikan formal TK Aisyiyah I Banyuwangi, SD Negeri 1 Kebalenan (2002-2008), SMP Negeri 1 Banyuwangi (2008-2011) dan SMA Negeri 1 Glagah (2011-2014). Penulis melanjutkan studi kuliah program sarjana di Jurusan Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Sistem Operasi (2016 dan 2017), Jaringan Komputer(2016). Selama menempuh perkuliahan penulis juga aktif di kegiatan organisasi dan kepanitiaan diantaranya menjadi staff Departemen Pengembangan Profesi HMTC ITS, wakil dua Bendahara Schematics 2015, Bendahara Schematics 2015 dan panitia Pemusatan Latihan Nasional 2 TOKI 2017 di ITS. Penulis dapat dihubungi melalui surel di desyrahmi09@gmail.com.